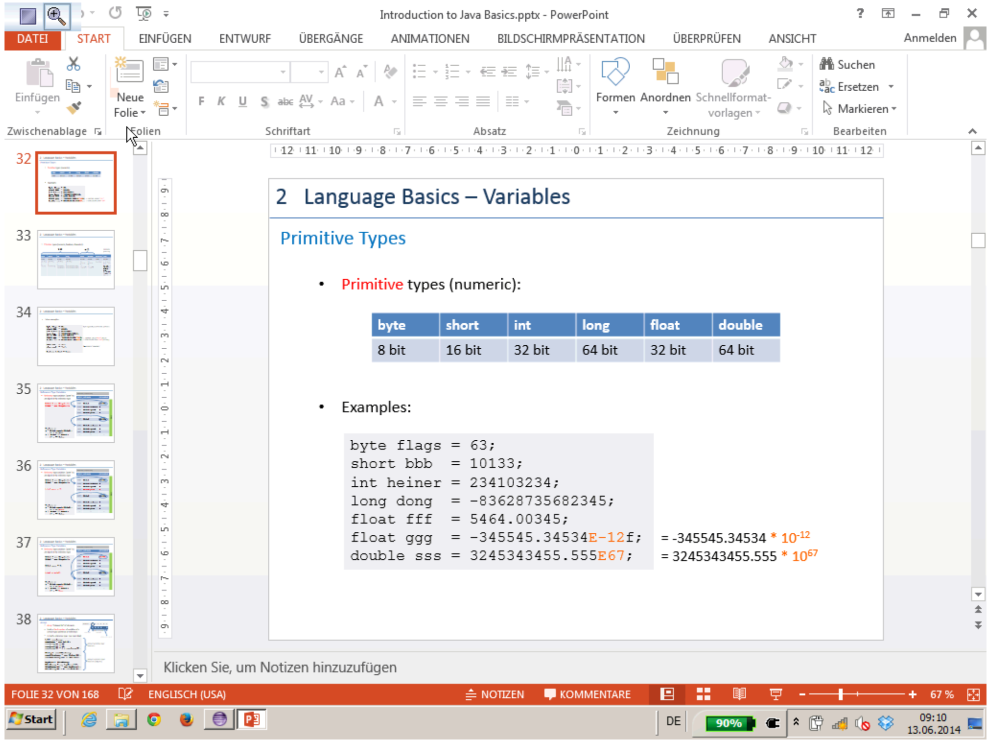
Script **generated by TTT**

Title: groh: profile1 (13.06.2014)

Date: Fri Jun 13 09:10:04 CEST 2014

Duration: 99:13 min

Pages: 98



2 Language Basics – Variables

Primitive Types

- Primitive types (numeric):

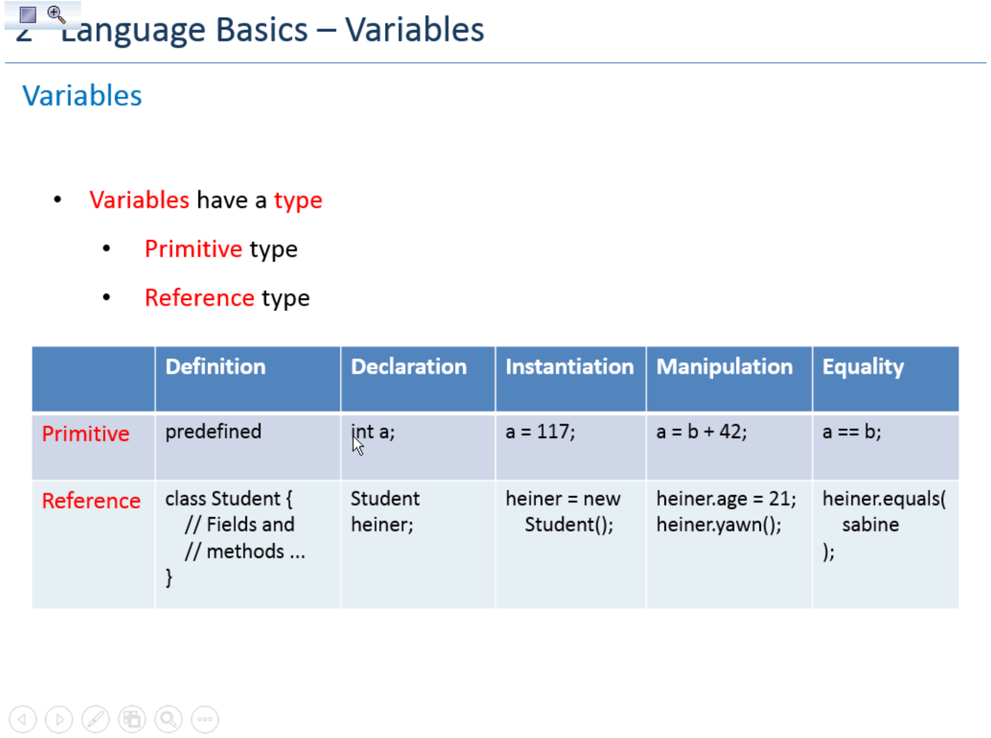
byte	short	int	long	float	double
8 bit	16 bit	32 bit	64 bit	32 bit	64 bit

- Examples:

```
byte flags = 63;
short bbb  = 10133;
int heiner = 234103234;
long dong  = -83628735682345;
float fff  = 5464.00345;
float ggg  = -345545.34534E-12f;
double sss = 3245343455.555E67;
```

= -345545.34534 * 10^{-12}

= 3245343455.555 * 10^{67}



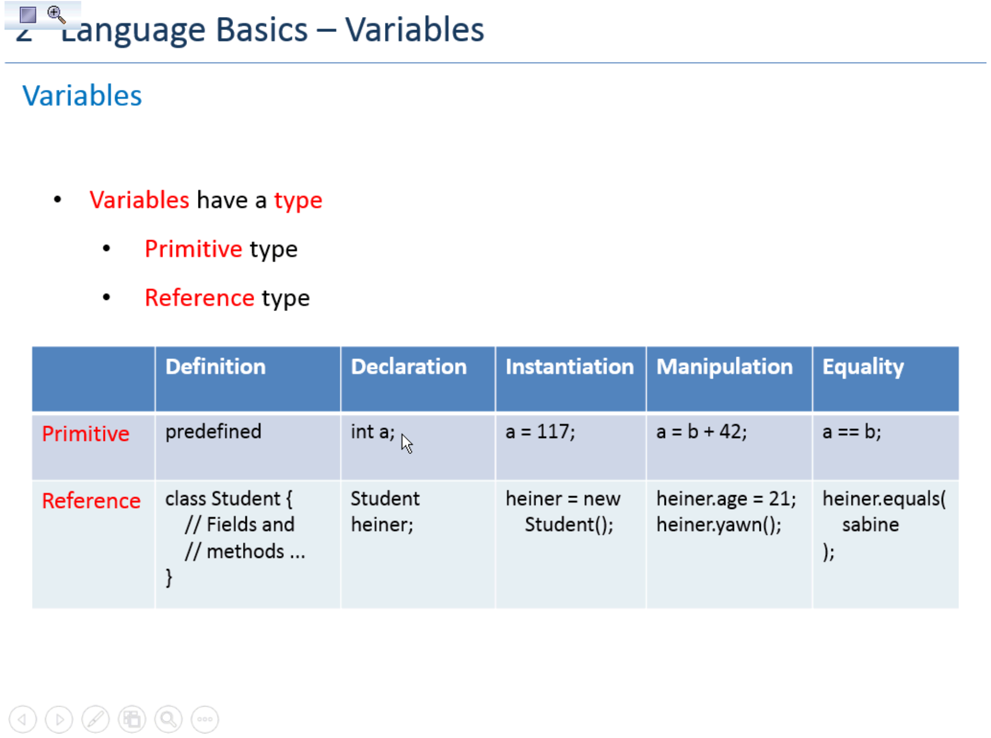
2 Language Basics – Variables

Variables

- Variables have a type
  - Primitive type
  - Reference type

	Definition	Declaration	Instantiation	Manipulation	Equality
Primitive	predefined	int a;	a = 117;	a = b + 42;	a == b;

Reference	class Student {
	// Fields and
	// methods ...
	}
		Student heiner;	heiner = new Student();	heiner.age = 21;
				heiner.yawn();
					heiner.equals(sabine);



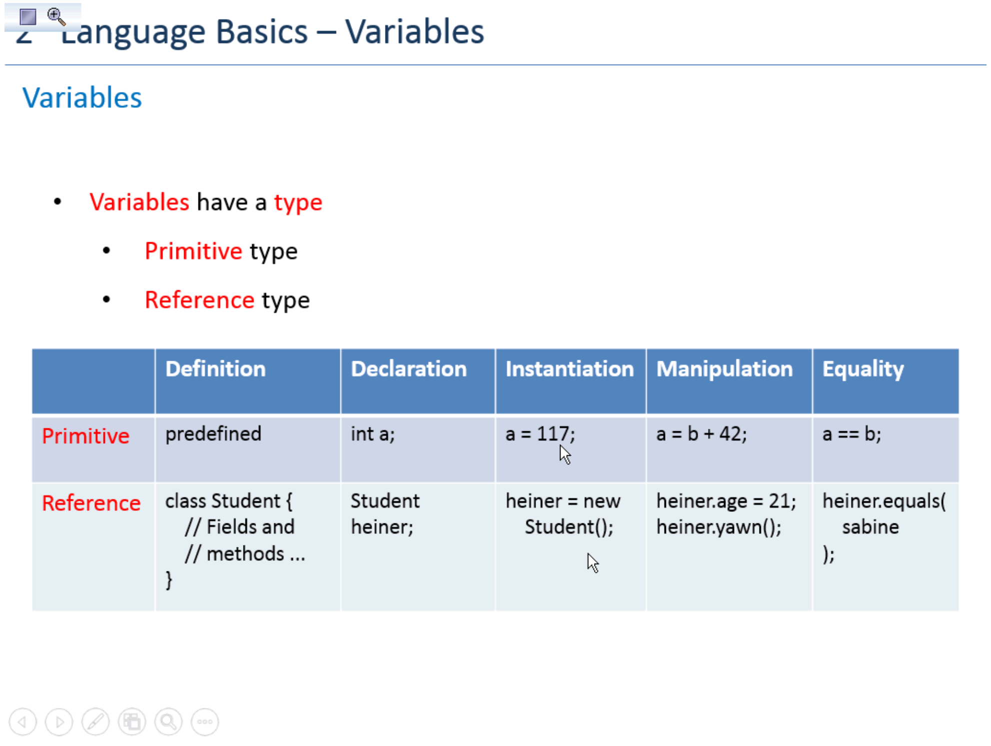
2 Language Basics – Variables

Variables

- Variables have a type
  - Primitive type
  - Reference type

	Definition	Declaration	Instantiation	Manipulation	Equality
Primitive	predefined	int a;	a = 117;	a = b + 42;	a == b;

Reference	class Student {
	// Fields and
	// methods ...
	}
		Student heiner;	heiner = new Student();	heiner.age = 21;
				heiner.yawn();
					heiner.equals(sabine);



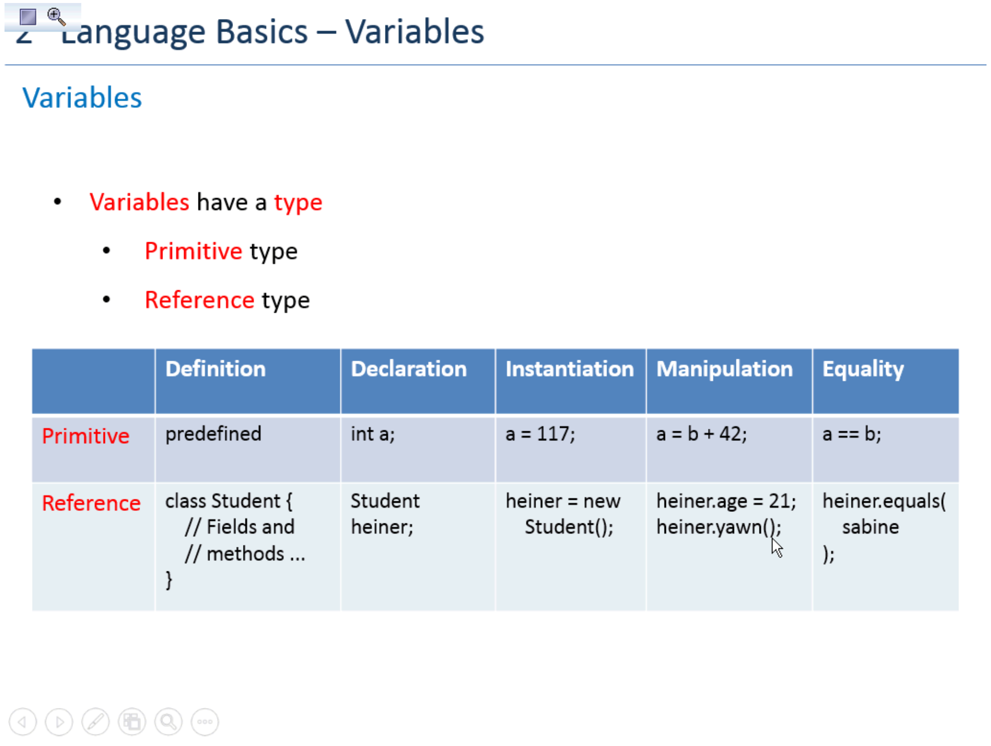
2 Language Basics – Variables

Variables

- Variables have a type
  - Primitive type
  - Reference type

	Definition	Declaration	Instantiation	Manipulation	Equality
Primitive	predefined	int a;	a = 117;	a = b + 42;	a == b;

Reference	class Student {
	// Fields and
	// methods ...
	}
		Student heiner;	heiner = new Student();	heiner.age = 21;
				heiner.yawn();
					heiner.equals(
					sabine
					);



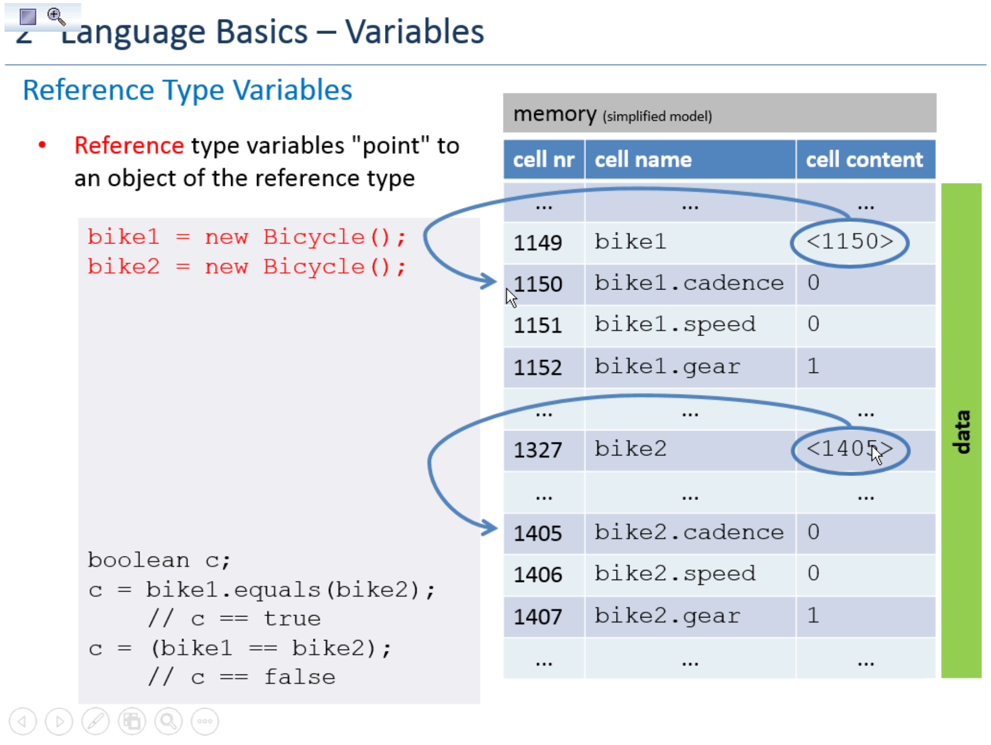
2 Language Basics – Variables

Variables

- Variables have a type
  - Primitive type
  - Reference type

	Definition	Declaration	Instantiation	Manipulation	Equality
Primitive	predefined	int a;	a = 117;	a = b + 42;	a == b;

Reference	class Student {
	// Fields and
	// methods ...
	}
		Student heiner;	heiner = new Student();	heiner.age = 21;
				heiner.yawn();
					heiner.equals(
					sabine
					);



2 Language Basics – Variables

Reference Type Variables

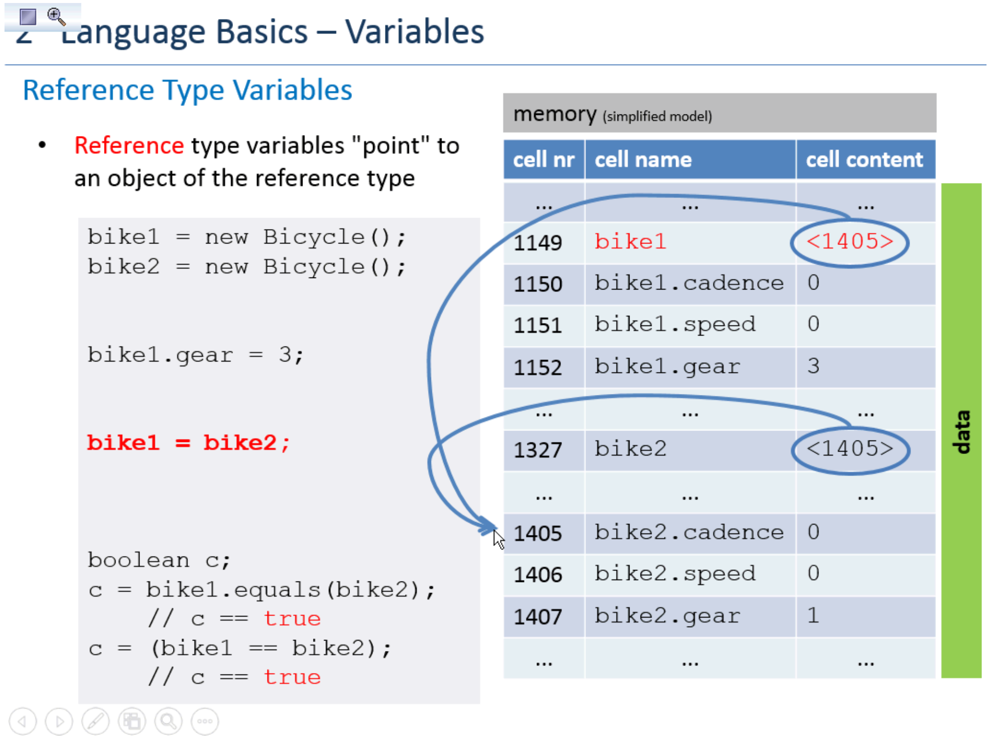
- Reference type variables "point" to an object of the reference type

```
bike1 = new Bicycle();
bike2 = new Bicycle();




boolean c;
c = bike1.equals(bike2);
    // c == true
c = (bike1 == bike2);
    // c == false
```

memory (simplified model)

cell nr	cell name	cell content
...	...	...
1149	bike1	<1150>
1150	bike1.cadence	0
1151	bike1.speed	0
1152	bike1.gear	1
...	...	...
1327	bike2	<1405>
...	...	...
1405	bike2.cadence	0
1406	bike2.speed	0
1407	bike2.gear	1
...	...	...

data



2 Language Basics – Variables

Reference Type Variables

- Reference type variables "point" to an object of the reference type

```
bike1 = new Bicycle();
bike2 = new Bicycle();


bike1.gear = 3;


bike1 = bike2;


boolean c;
c = bike1.equals(bike2);
    // c == true
c = (bike1 == bike2);
    // c == true
```

memory (simplified model)

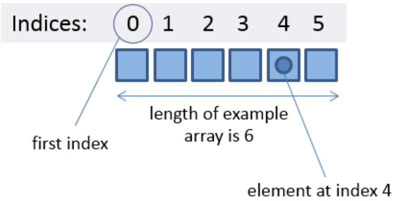
cell nr	cell name	cell content
...	...	...
1149	bike1	<1405>
1150	bike1.cadence	0
1151	bike1.speed	0
1152	bike1.gear	3
...	...	...
1327	bike2	<1405>
...	...	...
1405	bike2.cadence	0
1406	bike2.speed	0
1407	bike2.gear	1
...	...	...

data

2 Language Basics – Variables

Arrays

- Array: "Indexed list" of elements
- Holds a fixed number of variables of a certain type (primitive or reference)
- Is itself a reference type (see next slide)



```
int[] someArray;
someArray = new int[6];
someArray[0] = 23;
someArray[1] = 12;
someArray[5] = 4 + someArray[2];

String[] someOtherArray;
someOtherArray = new String[30];
someOtherArray[17] = "bla bla";

AnyClass[] thirdArray;
thirdArray = new AnyClass[45];
thirdArray[44] = new AnyClass();
thirdArray[22 * 2].someMethod();
```

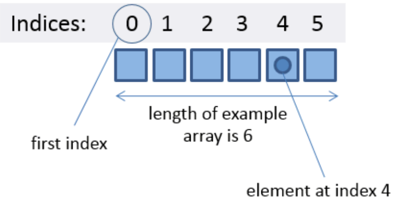
array of *primitive type* elements

array of *reference type* elements (objects)

2 Language Basics – Variables

Arrays

- Array: "Indexed list" of elements
- Holds a fixed number of variables of a certain type (primitive or reference)
- Is itself a reference type (see next slide)



```
int[] someArray;
someArray = new int[6];
someArray[0] = 23;
someArray[1] = 12;
someArray[5] = 4 + someArray[2];

String[] someOtherArray;
someOtherArray = new String[30];
someOtherArray[17] = "bla bla";

AnyClass[] thirdArray;
thirdArray = new AnyClass[45];
thirdArray[44] = new AnyClass();
thirdArray[22 * 2].someMethod();
```

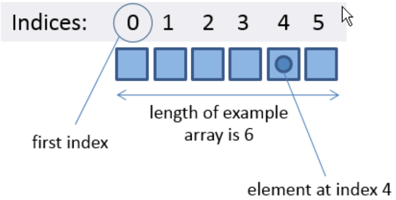
array of *primitive type* elements

array of *reference type* elements (objects)

2 Language Basics – Variables

Arrays

- Array: "Indexed list" of elements
- Holds a fixed number of variables of a certain type (primitive or reference)
- Is itself a reference type (see next slide)



```
int[] someArray;
someArray = new int[6];
someArray[0] = 23;
someArray[1] = 12;
someArray[5] = 4 + someArray[2];

String[] someOtherArray;
someOtherArray = new String[30];
someOtherArray[17] = "bla bla";

AnyClass[] thirdArray;
thirdArray = new AnyClass[45];
thirdArray[44] = new AnyClass();
thirdArray[22 * 2].someMethod();
```

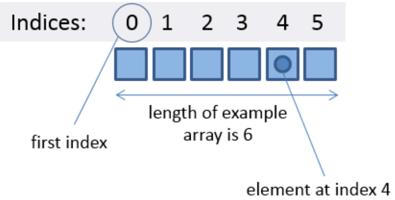
array of *primitive type* elements

array of *reference type* elements (objects)

2 Language Basics – Variables

Arrays

- Array: "Indexed list" of elements
- Holds a fixed number of variables of a certain type (primitive or reference)
- Is itself a reference type (see next slide)



```
int[] someArray;
someArray = new int[6];
someArray[0] = 23;
someArray[1] = 12;
someArray[5] = 4 + someArray[2];

String[] someOtherArray;
someOtherArray = new String[30];
someOtherArray[17] = "bla bla";

AnyClass[] thirdArray;
thirdArray = new AnyClass[45];
thirdArray[44] = new AnyClass();
thirdArray[22 * 2].someMethod();
```

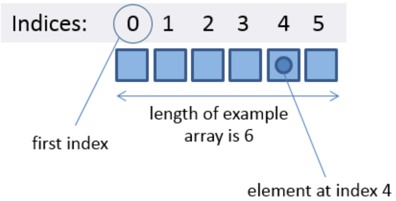
array of *primitive type* elements

array of *reference type* elements (objects)

2 Language Basics – Variables

Arrays

- Array: "Indexed list" of elements
- Holds a fixed number of variables of a certain type (primitive or reference)
- Is itself a reference type (see next slide)



Indices: 0 1 2 3 4 5

first index

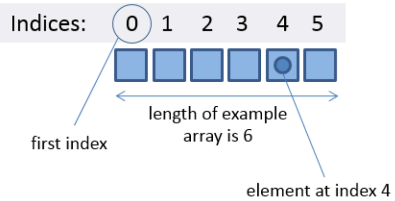
length of example array is 6

element at index 4

```
int[] someArray;
someArray = new int[6];
someArray[0] = 23;
someArray[1] = 12;
someArray[5] = 4 + someArray[2];

String[] someOtherArray;
someOtherArray = new String[30];
someOtherArray[17] = "bla bla";

AnyClass[] thirdArray;
thirdArray = new AnyClass[45];
thirdArray[44] = new AnyClass();
thirdArray[22 * 2].someMethod();
```

array of *primitive type* elements

array of *reference type* elements (objects)



2 Language Basics – Variables

Arrays

- Array: "Indexed list" of elements
- Holds a fixed number of variables of a certain type (primitive or reference)
- Is itself a reference type (see next slide)

Indices: 0 1 2 3 4 5

first index

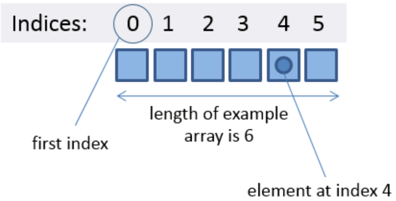
length of example array is 6

element at index 4

```
int[] someArray;
someArray = new int[6];
someArray[0] = 23;
someArray[1] = 12;
someArray[5] = 4 + someArray[2];

String[] someOtherArray;
someOtherArray = new String[30];
someOtherArray[17] = "bla bla";

AnyClass[] thirdArray;
thirdArray = new AnyClass[45];
thirdArray[44] = new AnyClass();
thirdArray[22 * 2].someMethod();
```

array of *primitive type* elements

array of *reference type* elements (objects)



2 Language Basics – Variables

Arrays

- Array: "Indexed list" of elements
- Holds a fixed number of variables of a certain type (primitive or reference)
- Is itself a reference type (see next slide)

Indices: 0 1 2 3 4 5

first index

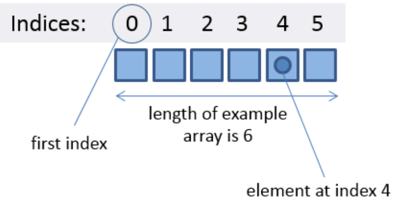
length of example array is 6

element at index 4

```
int[] someArray;
someArray = new int[6];
someArray[0] = 23;
someArray[1] = 12;
someArray[5] = 4 + someArray[2];

String[] someOtherArray;
someOtherArray = new String[30];
someOtherArray[17] = "bla bla";

AnyClass[] thirdArray;
thirdArray = new AnyClass[45];
thirdArray[44] = new AnyClass();
thirdArray[22 * 2].someMethod();
```

array of *primitive type* elements

array of *reference type* elements (objects)



2 Language Basics – Variables

Arrays

- Array: "Indexed list" of elements
- Holds a fixed number of variables of a certain type (primitive or reference)
- Is itself a reference type (see next slide)

Indices: 0 1 2 3 4 5

first index

length of example array is 6

element at index 4

```
int[] someArray;
someArray = new int[6];
someArray[0] = 23;
someArray[1] = 12;
someArray[5] = 4 + someArray[2];

String[] someOtherArray;
someOtherArray = new String[30];
someOtherArray[17] = "bla bla";

AnyClass[] thirdArray;
thirdArray = new AnyClass[45];
thirdArray[44] = new AnyClass();
thirdArray[22 * 2].someMethod();
```

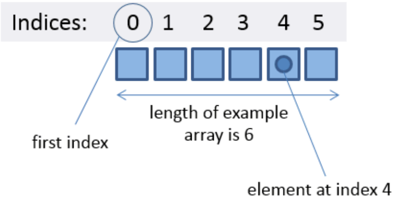
array of *primitive type* elements

array of *reference type* elements (objects)

Language Basics – Variables

Arrays

- Array: "Indexed list" of elements
- Holds a fixed number of variables of a certain type (primitive or reference)
- Is itself a reference type (see next slide)



```
int[] someArray;
someArray = new int[6];
someArray[0] = 23;
someArray[1] = 12;
someArray[5] = 4 + someArray[2];
```

array of *primitive type* elements

```
String[] someOtherArray;
someOtherArray = new String[30];
someOtherArray[17] = "bla bla";

AnyClass[] thirdArray;
thirdArray = new AnyClass[45];
thirdArray[44] = new AnyClass();
thirdArray[22 * 2].someMethod();
```

array of *reference type* elements (objects)

Language Basics – Variables

Arrays

- Array is itself a reference type:

```
int[] someArray = new int[3];
int[] anotherArray = new int[3];

someArray[2] = 7;
anotherArray[1] = 8;
```

memory (simplified model)

cell nr	cell name	cell content
...	...	...
1149	someArray	<1150>
1150		0
1151		0
1152		7
...	...	...
1327	anotherArray	<1328>
1328		0
1329		8
1330		0
...	...	...

Language Basics – Variables

Arrays

- Array is itself a reference type:

```
int[] someArray = new int[3];
int[] anotherArray = new int[3];

someArray[2] = 7;
anotherArray[1] = 8;

someArray = anotherArray;

boolean b = (someArray[1] == 8);
// b == true
```

- Length property:

```
int l = someArray.length;
// l == 3
```

memory (simplified model)

cell nr	cell name	cell content
...	...	...
1149	someArray	<1328>
1150		0
1151		0
1152		7
...	...	...
1327	anotherArray	<1328>
1328		0
1329		8
1330		0
...	...	...

Language Basics – Variables

Arrays

- Array is itself a reference type:

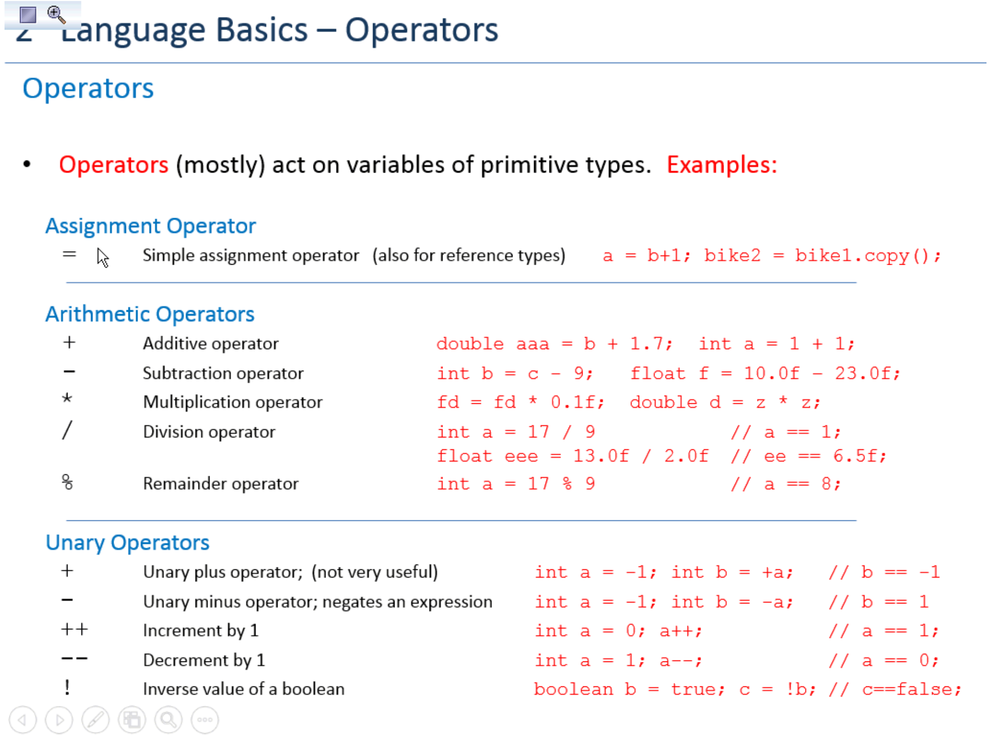
```
int[] someArray = new int[3];
int[] anotherArray = new int[3];

someArray[2] = 7;
anotherArray[1] = 8;

someArray = anotherArray;

boolean b = (someArray[1] == 8);
// b == true
```

- Length property:

```
int l = someArray.length;
// l == 3
```

memory (simplified model)

cell nr	cell name	cell content
...	...	...
1149	someArray	<1328>
1150		0
1151		0
1152		7
...	...	...
1327	anotherArray	<1328>
1328		0
1329		8
1330		0
...	...	...



2 Language Basics – Operators

Operators

- Operators (mostly) act on variables of primitive types. Examples:

Assignment Operator

=	Simple assignment operator (also for reference types)	`a = b+1; bike2 = bike1.copy();`

Arithmetic Operators

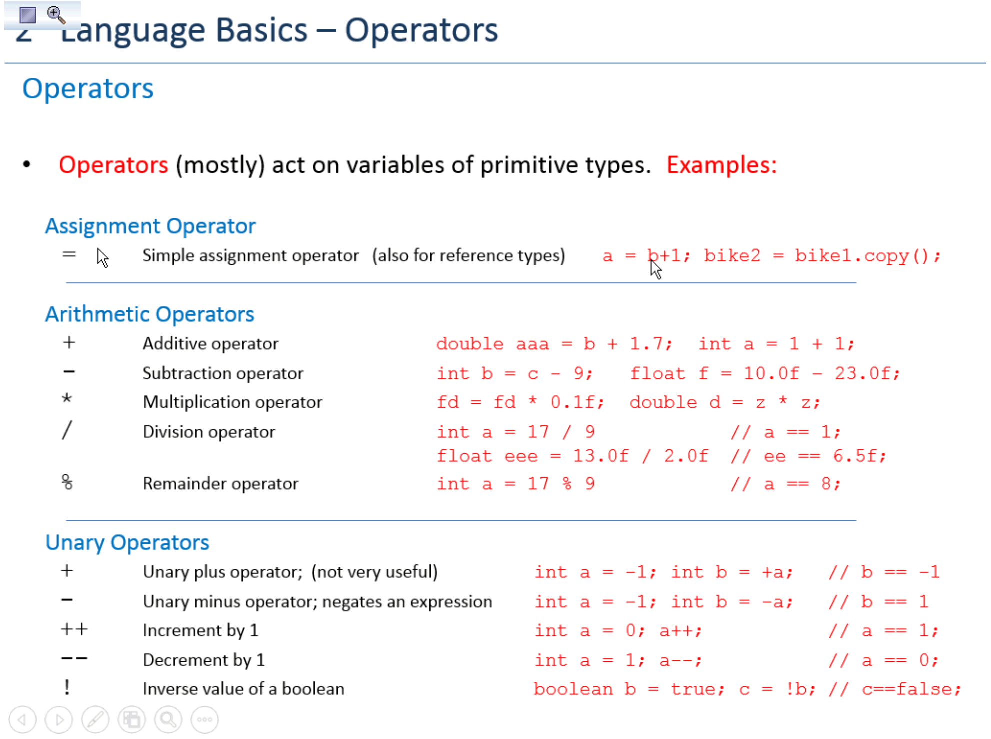
+	Additive operator	`double aaa = b + 1.7;  int a = 1 + 1;`
–	Subtraction operator	`int b = c - 9;   float f = 10.0f - 23.0f;`
*	Multiplication operator	`fd = fd * 0.1f;  double d = z * z;`

/	Division operator	`int a = 17 / 9            // a == 1;`
		`float eee = 13.0f / 2.0f  // ee == 6.5f;`

%	Remainder operator	`int a = 17 % 9            // a == 8;`

Unary Operators

+	Unary plus operator; (not very useful)	`int a = -1; int b = +a;   // b == -1`
–	Unary minus operator; negates an expression	`int a = -1; int b = -a;   // b == 1`
++	Increment by 1	`int a = 0; a++;            // a == 1;`
––	Decrement by 1	`int a = 1; a--;            // a == 0;`
!	Inverse value of a boolean	`boolean b = true; c = !b; // c==false;`



2 Language Basics – Operators

Operators

- Operators (mostly) act on variables of primitive types. Examples:

Assignment Operator

=	Simple assignment operator (also for reference types)	`a = b+1; bike2 = bike1.copy();`

Arithmetic Operators

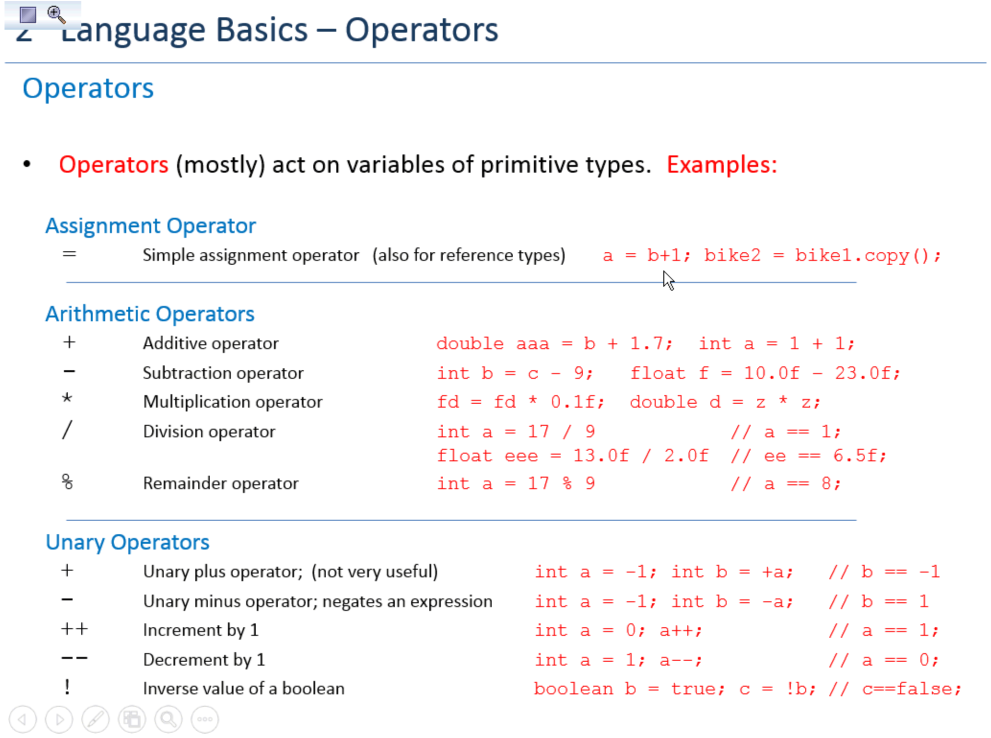
+	Additive operator	`double aaa = b + 1.7;  int a = 1 + 1;`
–	Subtraction operator	`int b = c - 9;   float f = 10.0f - 23.0f;`
*	Multiplication operator	`fd = fd * 0.1f;  double d = z * z;`

/	Division operator	`int a = 17 / 9            // a == 1;`
		`float eee = 13.0f / 2.0f  // ee == 6.5f;`

%	Remainder operator	`int a = 17 % 9            // a == 8;`

Unary Operators

+	Unary plus operator; (not very useful)	`int a = -1; int b = +a;   // b == -1`
–	Unary minus operator; negates an expression	`int a = -1; int b = -a;   // b == 1`
++	Increment by 1	`int a = 0; a++;            // a == 1;`
––	Decrement by 1	`int a = 1; a--;            // a == 0;`
!	Inverse value of a boolean	`boolean b = true; c = !b; // c==false;`



2 Language Basics – Operators

Operators

- Operators (mostly) act on variables of primitive types. Examples:

Assignment Operator

=	Simple assignment operator (also for reference types)	`a = b+1; bike2 = bike1.copy();`

Arithmetic Operators

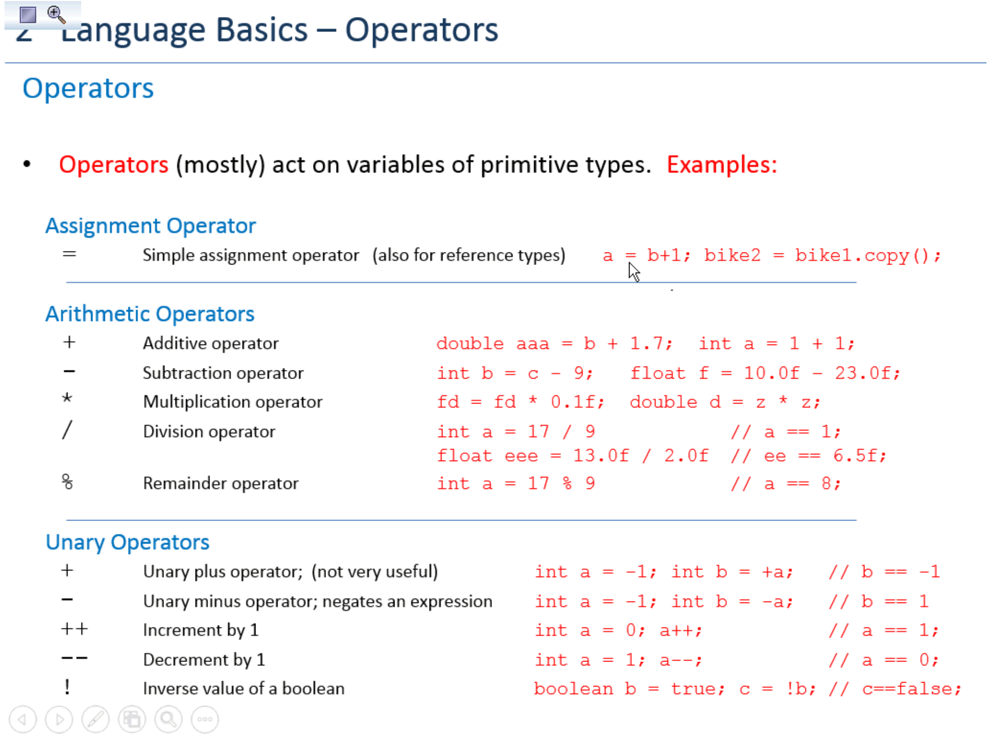
+	Additive operator	`double aaa = b + 1.7;  int a = 1 + 1;`
–	Subtraction operator	`int b = c - 9;   float f = 10.0f - 23.0f;`
*	Multiplication operator	`fd = fd * 0.1f;  double d = z * z;`

/	Division operator	`int a = 17 / 9            // a == 1;`
		`float eee = 13.0f / 2.0f  // ee == 6.5f;`

%	Remainder operator	`int a = 17 % 9            // a == 8;`

Unary Operators

+	Unary plus operator; (not very useful)	`int a = -1; int b = +a;   // b == -1`
–	Unary minus operator; negates an expression	`int a = -1; int b = -a;   // b == 1`
++	Increment by 1	`int a = 0; a++;            // a == 1;`
––	Decrement by 1	`int a = 1; a--;            // a == 0;`
!	Inverse value of a boolean	`boolean b = true; c = !b; // c==false;`



2 Language Basics – Operators

Operators

- Operators (mostly) act on variables of primitive types. Examples:

Assignment Operator

=	Simple assignment operator (also for reference types)	`a = b+1; bike2 = bike1.copy();`

Arithmetic Operators

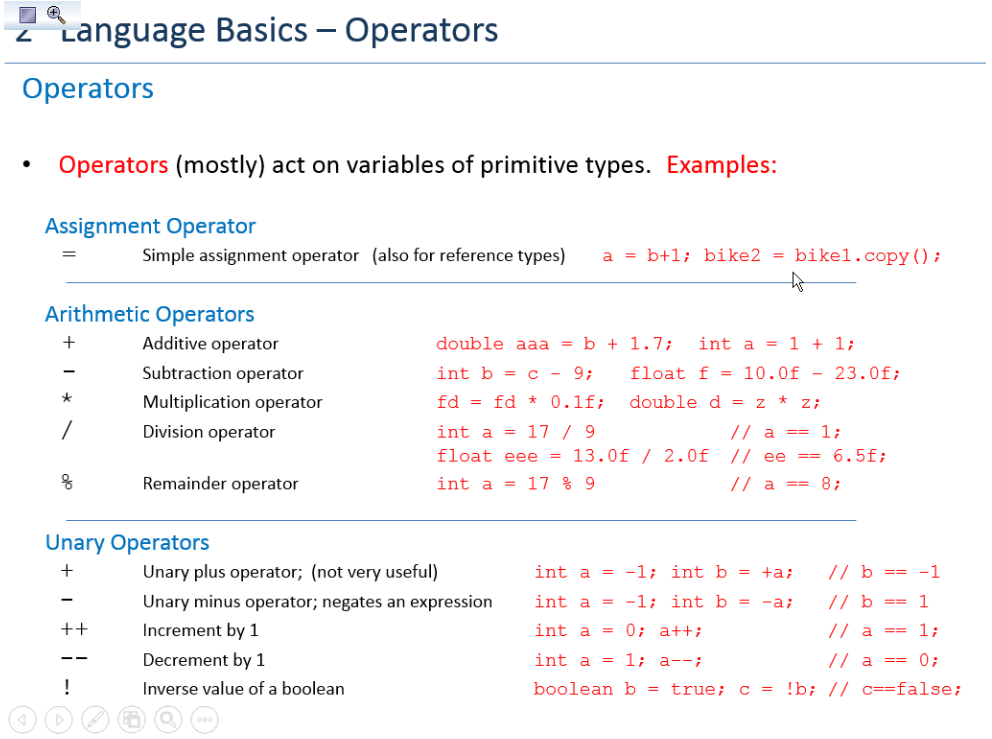
+	Additive operator	`double aaa = b + 1.7;  int a = 1 + 1;`
–	Subtraction operator	`int b = c - 9;   float f = 10.0f - 23.0f;`
*	Multiplication operator	`fd = fd * 0.1f;  double d = z * z;`

/	Division operator	`int a = 17 / 9            // a == 1;`
		`float eee = 13.0f / 2.0f  // ee == 6.5f;`

%	Remainder operator	`int a = 17 % 9            // a == 8;`

Unary Operators

+	Unary plus operator; (not very useful)	`int a = -1; int b = +a;   // b == -1`
–	Unary minus operator; negates an expression	`int a = -1; int b = -a;   // b == 1`
++	Increment by 1	`int a = 0; a++;            // a == 1;`
––	Decrement by 1	`int a = 1; a--;            // a == 0;`
!	Inverse value of a boolean	`boolean b = true; c = !b; // c==false;`



2 Language Basics – Operators

Operators

- Operators (mostly) act on variables of primitive types. Examples:

Assignment Operator

=	Simple assignment operator (also for reference types)	`a = b+1; bike2 = bike1.copy();`

Arithmetic Operators

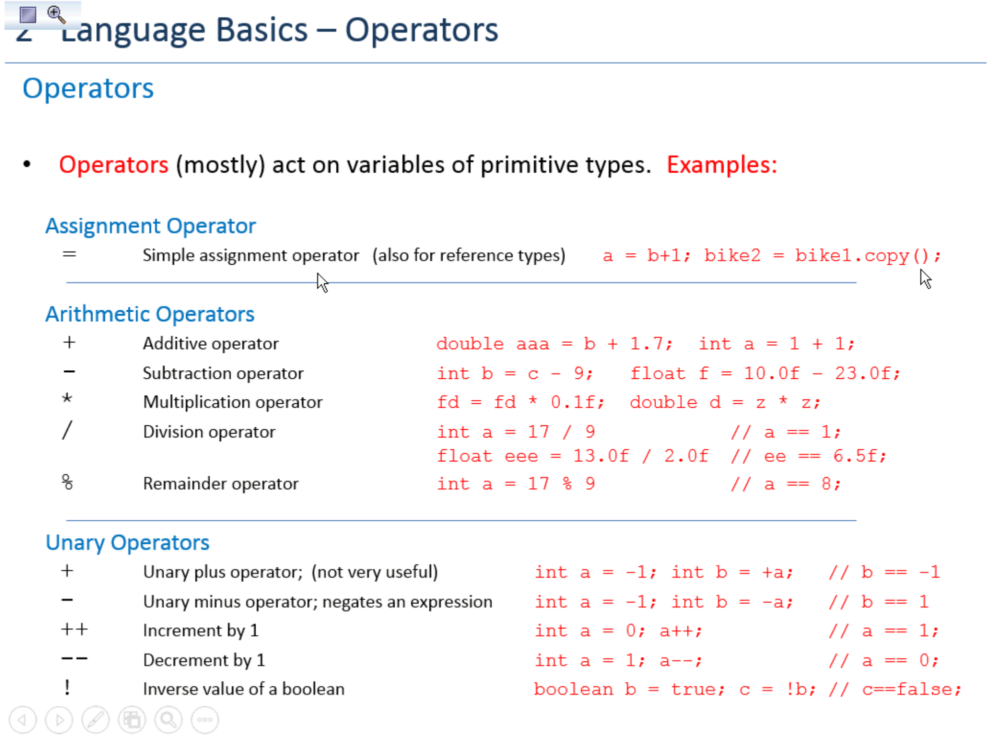
+	Additive operator	`double aaa = b + 1.7; int a = 1 + 1;`
–	Subtraction operator	`int b = c - 9; float f = 10.0f - 23.0f;`
*	Multiplication operator	`fd = fd * 0.1f; double d = z * z;`

/	Division operator	`int a = 17 / 9 // a == 1;`
		`float eee = 13.0f / 2.0f // ee == 6.5f;`

%	Remainder operator	`int a = 17 % 9 // a == 8;`

Unary Operators

+	Unary plus operator; (not very useful)	`int a = -1; int b = +a; // b == -1`
–	Unary minus operator; negates an expression	`int a = -1; int b = -a; // b == 1`
++	Increment by 1	`int a = 0; a++; // a == 1;`
--	Decrement by 1	`int a = 1; a--; // a == 0;`
!	Inverse value of a boolean	`boolean b = true; c = !b; // c==false;`



2 Language Basics – Operators

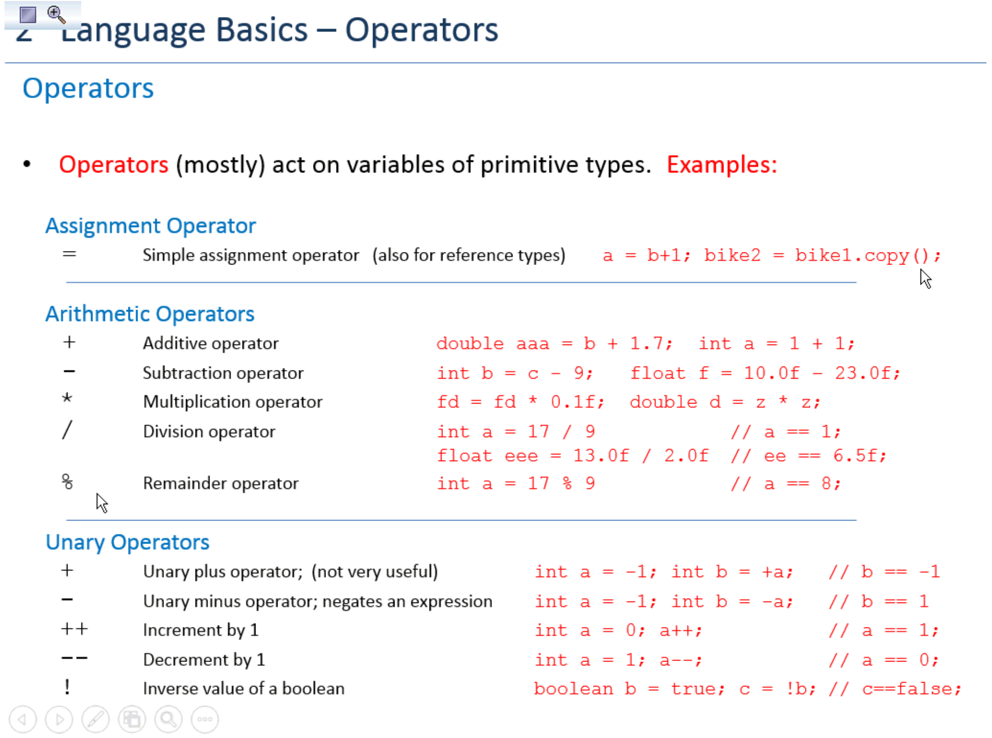
Operators

- Operators (mostly) act on variables of primitive types. Examples:

Assignment Operator
= Simple assignment operator (also for reference types) a = b+1; bike2 = bike1.copy();

Arithmetic Operators
+ Additive operator double aaa = b + 1.7; int a = 1 + 1;
– Subtraction operator int b = c - 9; float f = 10.0f - 23.0f;
* Multiplication operator fd = fd * 0.1f; double d = z * z;
/ Division operator int a = 17 / 9 // a == 1; float eee = 13.0f / 2.0f // ee == 6.5f;
% Remainder operator int a = 17 % 9 // a == 8;

Unary Operators
+ Unary plus operator; (not very useful) int a = -1; int b = +a; // b == -1
– Unary minus operator; negates an expression int a = -1; int b = -a; // b == 1
++ Increment by 1 int a = 0; a++; // a == 1;
-- Decrement by 1 int a = 1; a--; // a == 0;
! Inverse value of a boolean boolean b = true; c = !b; // c==false;



2 Language Basics – Operators

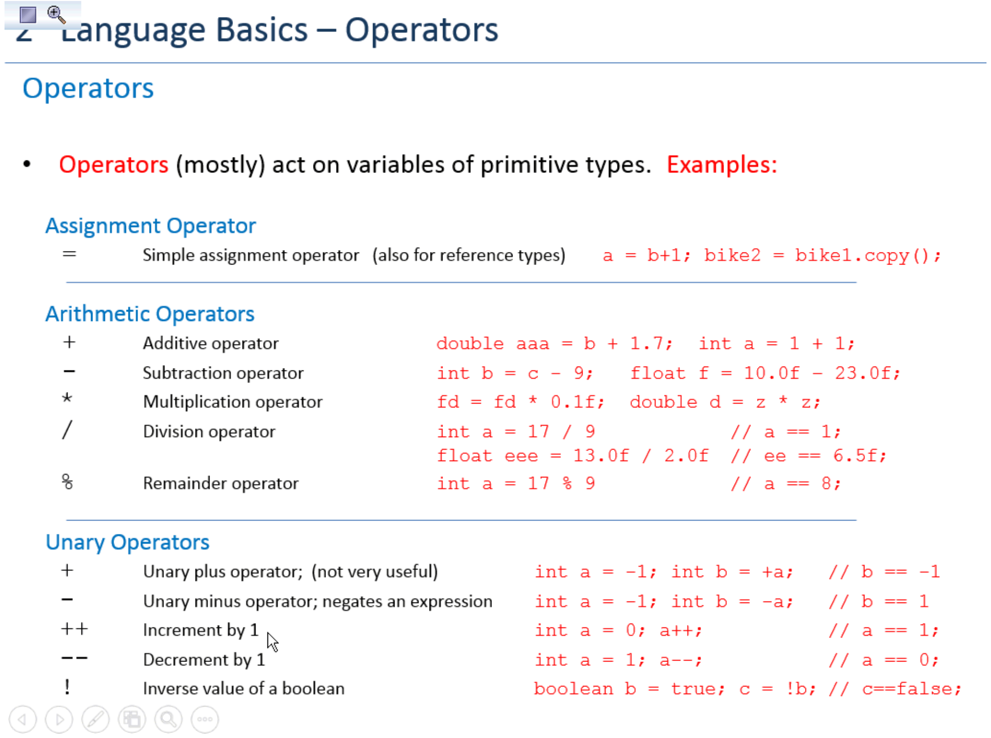
Operators

- Operators (mostly) act on variables of primitive types. Examples:

Assignment Operator
= Simple assignment operator (also for reference types) a = b+1; bike2 = bike1.copy();

Arithmetic Operators
+ Additive operator double aaa = b + 1.7; int a = 1 + 1;
– Subtraction operator int b = c - 9; float f = 10.0f - 23.0f;
* Multiplication operator fd = fd * 0.1f; double d = z * z;
/ Division operator int a = 17 / 9 // a == 1; float eee = 13.0f / 2.0f // ee == 6.5f;
% Remainder operator int a = 17 % 9 // a == 8;

Unary Operators
+ Unary plus operator; (not very useful) int a = -1; int b = +a; // b == -1
– Unary minus operator; negates an expression int a = -1; int b = -a; // b == 1
++ Increment by 1 int a = 0; a++; // a == 1;
-- Decrement by 1 int a = 1; a--; // a == 0;
! Inverse value of a boolean boolean b = true; c = !b; // c==false;



2 Language Basics – Operators

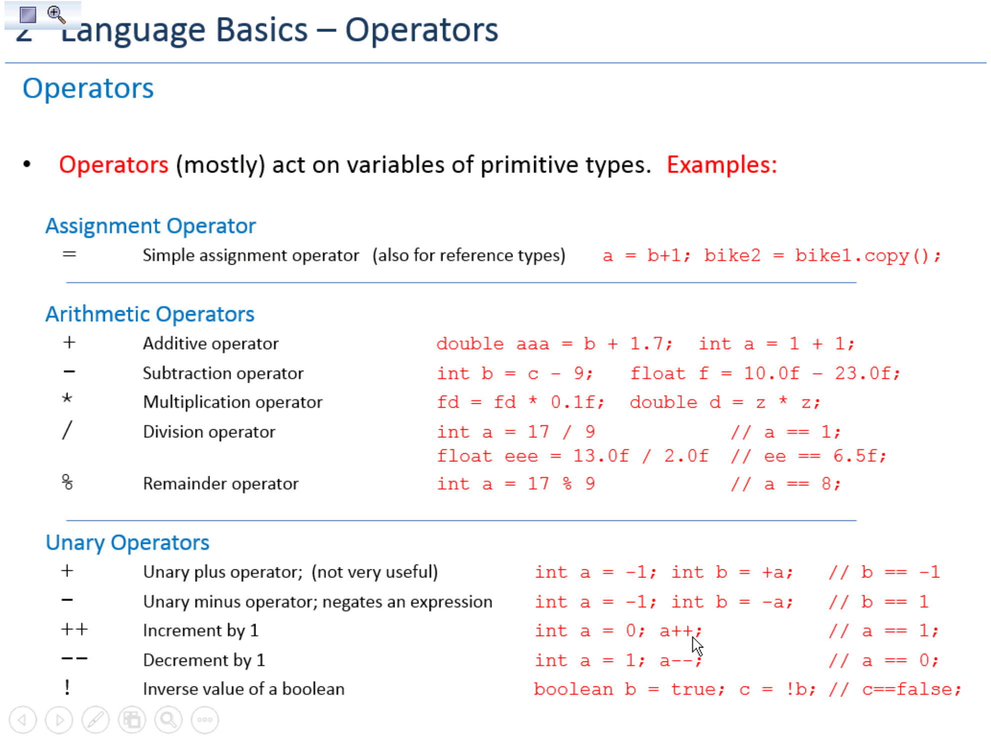
Operators

- Operators (mostly) act on variables of primitive types. Examples:

Assignment Operator
= Simple assignment operator (also for reference types) a = b+1; bike2 = bike1.copy();

Arithmetic Operators
+ Additive operator double aaa = b + 1.7; int a = 1 + 1;
– Subtraction operator int b = c - 9; float f = 10.0f - 23.0f;
* Multiplication operator fd = fd * 0.1f; double d = z * z;
/ Division operator int a = 17 / 9 // a == 1; float eee = 13.0f / 2.0f // ee == 6.5f;
% Remainder operator int a = 17 % 9 // a == 8;

Unary Operators
+ Unary plus operator; (not very useful) int a = -1; int b = +a; // b == -1
– Unary minus operator; negates an expression int a = -1; int b = -a; // b == 1
++ Increment by 1 int a = 0; a++; // a == 1;
-- Decrement by 1 int a = 1; a--; // a == 0;
! Inverse value of a boolean boolean b = true; c = !b; // c==false;



2 Language Basics – Operators

Operators

- Operators (mostly) act on variables of primitive types. Examples:

Assignment Operator

=	Simple assignment operator (also for reference types)	`a = b+1; bike2 = bike1.copy();`

Arithmetic Operators

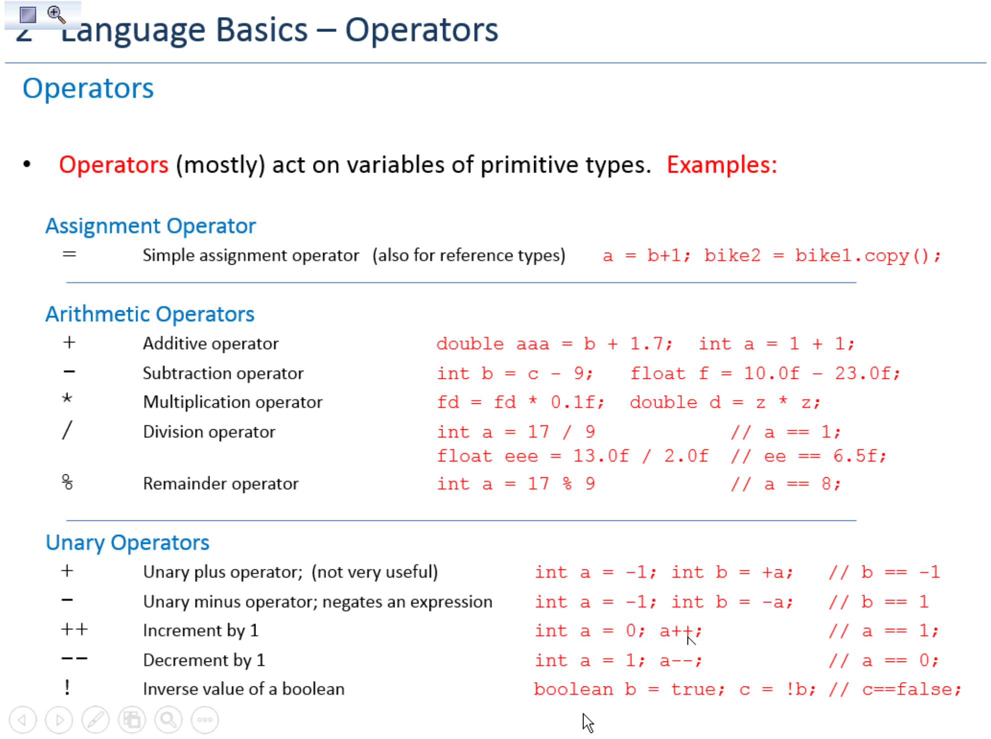
+	Additive operator	`double aaa = b + 1.7; int a = 1 + 1;`
–	Subtraction operator	`int b = c – 9; float f = 10.0f – 23.0f;`
*	Multiplication operator	`fd = fd * 0.1f; double d = z * z;`

/	Division operator	`int a = 17 / 9 // a == 1;`
		`float eee = 13.0f / 2.0f // ee == 6.5f;`

%	Remainder operator	`int a = 17 % 9 // a == 8;`

Unary Operators

+	Unary plus operator; (not very useful)	`int a = -1; int b = +a; // b == -1`
–	Unary minus operator; negates an expression	`int a = -1; int b = -a; // b == 1`
++	Increment by 1	`int a = 0; a++; // a == 1;`
--	Decrement by 1	`int a = 1; a--; // a == 0;`
!	Inverse value of a boolean	`boolean b = true; c = !b; // c==false;`



2 Language Basics – Operators

Operators

- Operators (mostly) act on variables of primitive types. Examples:

Assignment Operator

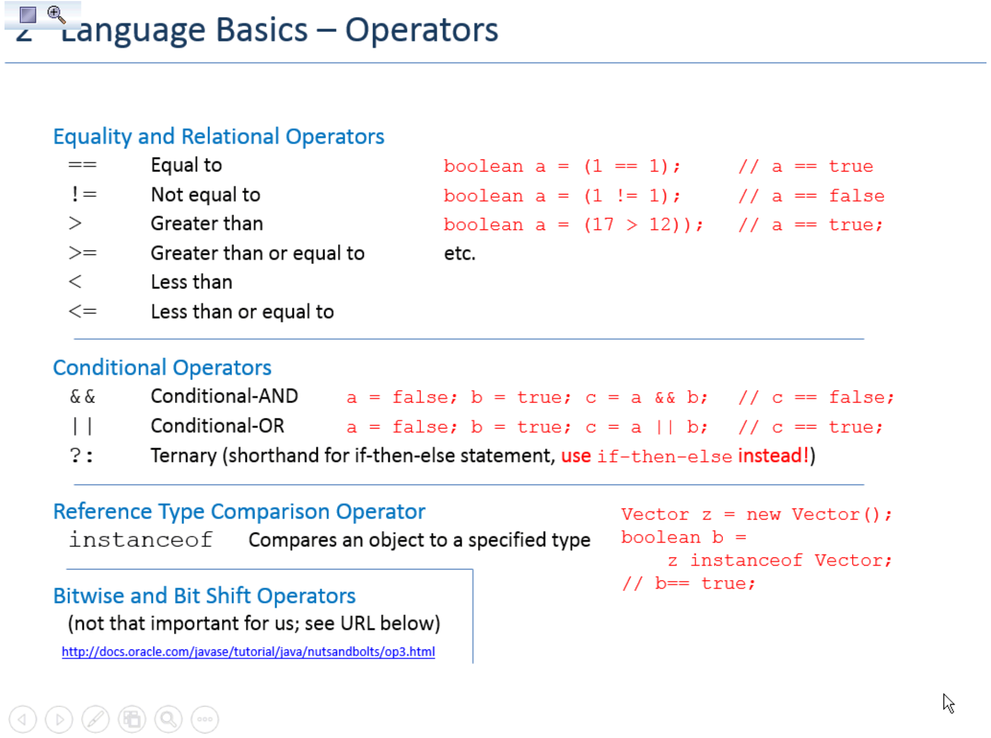
= Simple assignment operator (also for reference types) a = b+1; bike2 = bike1.copy();

Arithmetic Operators

+ Additive operator double aaa = b + 1.7; int a = 1 + 1;
– Subtraction operator int b = c – 9; float f = 10.0f – 23.0f;
* Multiplication operator fd = fd * 0.1f; double d = z * z;
/ Division operator int a = 17 / 9 // a == 1; float eee = 13.0f / 2.0f // ee == 6.5f;
% Remainder operator int a = 17 % 9 // a == 8;

Unary Operators

+ Unary plus operator; (not very useful) int a = -1; int b = +a; // b == -1
– Unary minus operator; negates an expression int a = -1; int b = -a; // b == 1
++ Increment by 1 int a = 0; a++; // a == 1;
-- Decrement by 1 int a = 1; a--; // a == 0;
! Inverse value of a boolean boolean b = true; c = !b; // c==false;



2 Language Basics – Operators

Equality and Relational Operators

==	Equal to	`boolean a = (1 == 1); // a == true`
!=	Not equal to	`boolean a = (1 != 1); // a == false`
>	Greater than	`boolean a = (17 > 12)); // a == true;`
>=	Greater than or equal to	etc.
<	Less than	
<=	Less than or equal to	

Conditional Operators

&& Conditional-AND `a = false; b = true; c = a && b; // c == false;`

|| Conditional-OR `a = false; b = true; c = a || b; // c == true;`

?: Ternary (shorthand for if-then-else statement, use `if-then-else` instead!)

Reference Type Comparison Operator

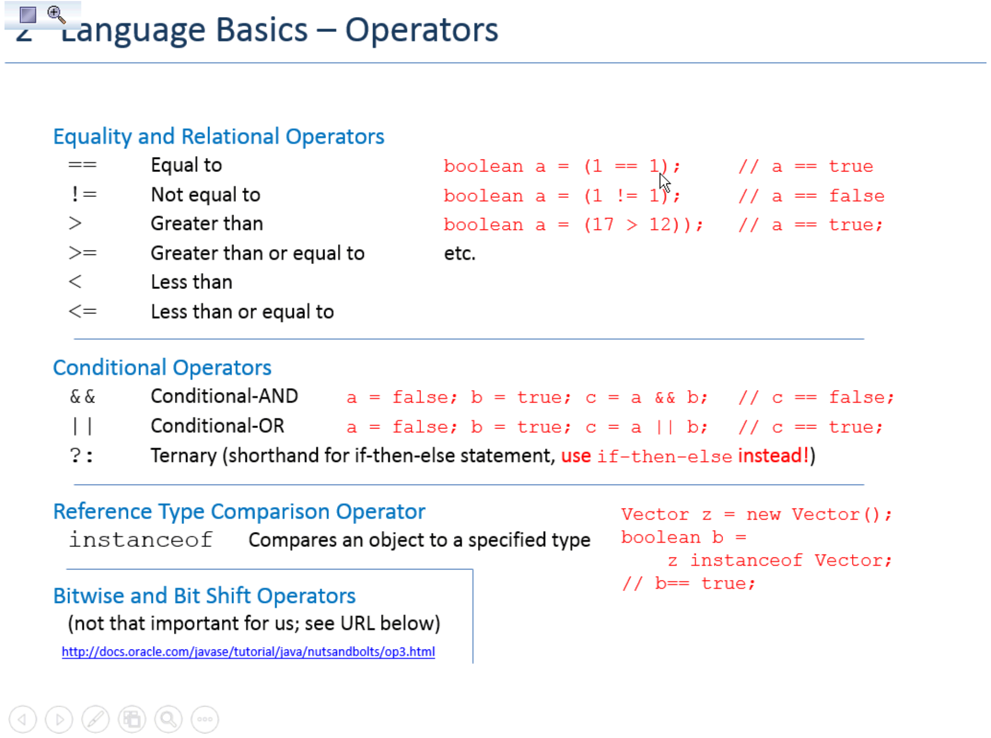
`instanceof` Compares an object to a specified type

```
Vector z = new Vector();
boolean b =
      z instanceof Vector;
// b== true;
```

Bitwise and Bit Shift Operators

(not that important for us; see URL below)

http://docs.oracle.com/javase/tutorial/java/nutsandbolts/op3.html



2 Language Basics – Operators

Equality and Relational Operators

== Equal to boolean a = (1 == 1); // a == true
!= Not equal to boolean a = (1 != 1); // a == false
> Greater than boolean a = (17 > 12)); // a == true;
>= Greater than or equal to etc.
< Less than
<= Less than or equal to

Conditional Operators

&& Conditional-AND a = false; b = true; c = a && b; // c == false;
|| Conditional-OR a = false; b = true; c = a || b; // c == true;
?: Ternary (shorthand for if-then-else statement, use if-then-else instead!)

Reference Type Comparison Operator

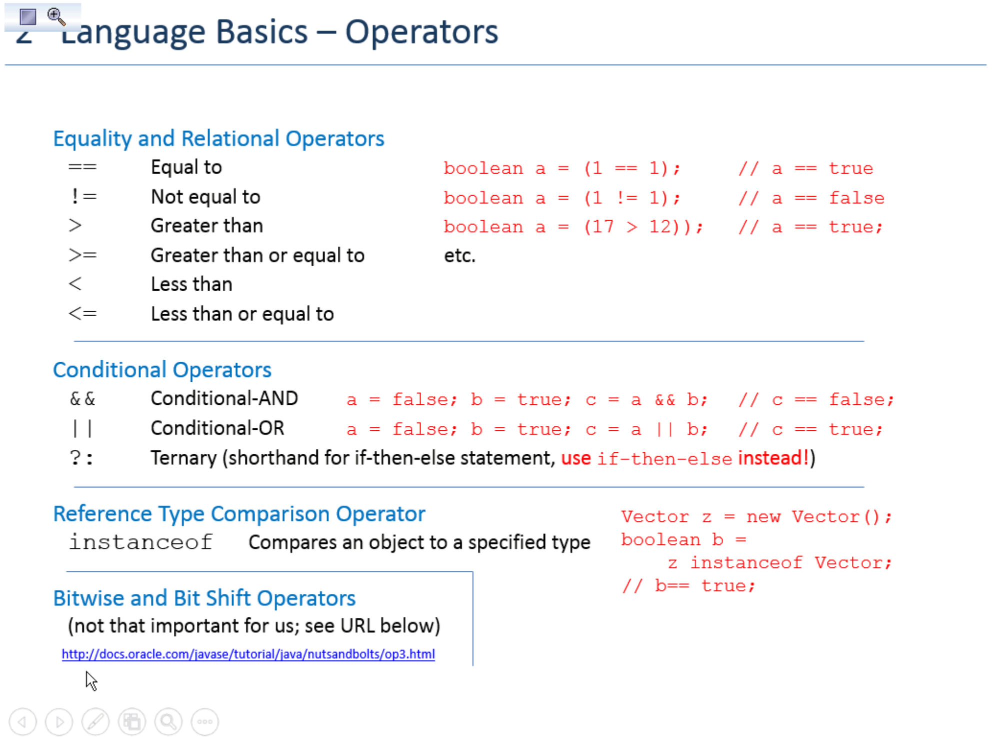
instanceof Compares an object to a specified type

Vector z = new Vector(); boolean b = z instanceof Vector; // b== true;

Bitwise and Bit Shift Operators

(not that important for us; see URL below)

http://docs.oracle.com/javase/tutorial/java/nutsandbolts/op3.html



2 Language Basics – Operators

Equality and Relational Operators

==	Equal to	`boolean a = (1 == 1);`	`// a == true`
!=	Not equal to	`boolean a = (1 != 1);`	`// a == false`
>	Greater than	`boolean a = (17 > 12));`	`// a == true;`
>=	Greater than or equal to	etc.	
<	Less than		
<=	Less than or equal to		

Conditional Operators

&&	Conditional-AND	`a = false; b = true; c = a && b;`	`// c == false;`
||	Conditional-OR	`a = false; b = true; c = a || b;`	`// c == true;`
?:	Ternary (shorthand for if-then-else statement, use `if-then-else` instead!)		

Reference Type Comparison Operator

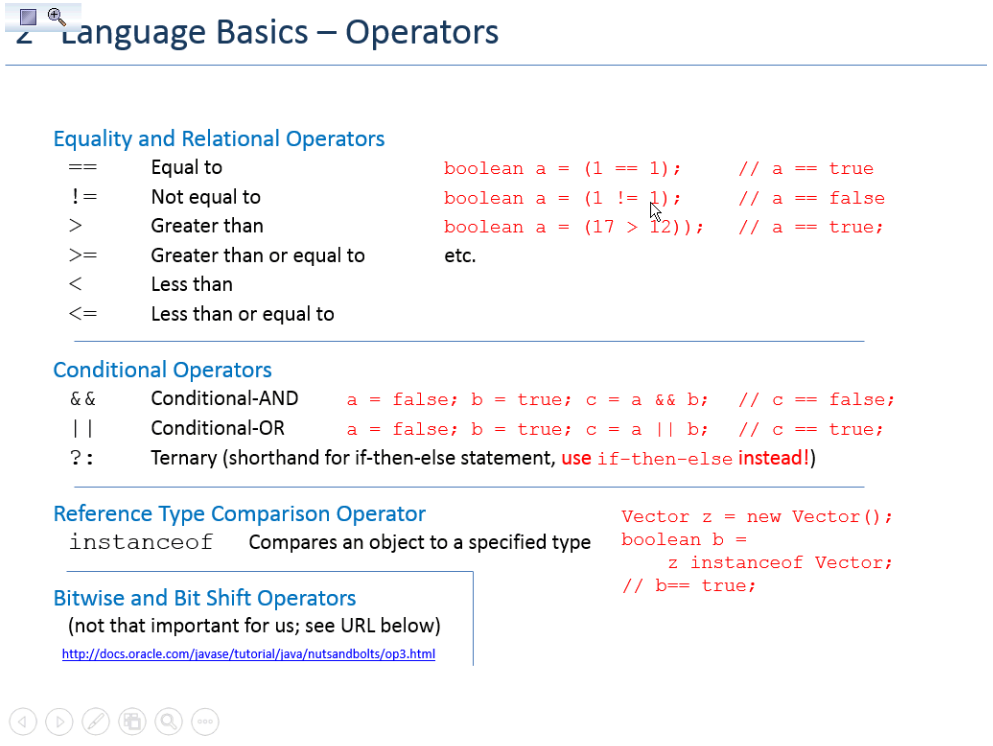
`instanceof` Compares an object to a specified type

```
Vector z = new Vector();
boolean b =
    z instanceof Vector;
// b== true;
```

Bitwise and Bit Shift Operators

(not that important for us; see URL below)

http://docs.oracle.com/javase/tutorial/java/nutsandbolts/op3.html



2 Language Basics – Operators

Equality and Relational Operators

==	Equal to	`boolean a = (1 == 1);`	`// a == true`
!=	Not equal to	`boolean a = (1 != 1);`	`// a == false`
>	Greater than	`boolean a = (17 > 12));`	`// a == true;`
>=	Greater than or equal to	etc.	
<	Less than		
<=	Less than or equal to		

Conditional Operators

&&	Conditional-AND	`a = false; b = true; c = a && b;`	`// c == false;`
||	Conditional-OR	`a = false; b = true; c = a || b;`	`// c == true;`
?:	Ternary (shorthand for if-then-else statement, use `if-then-else` instead!)		

Reference Type Comparison Operator

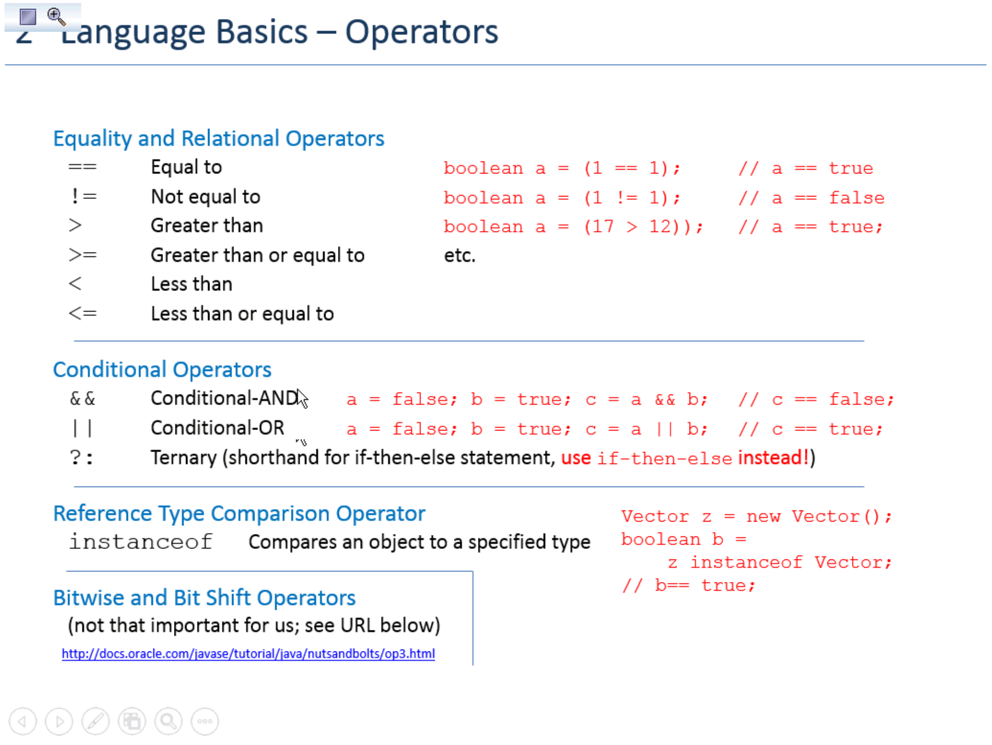
`instanceof` Compares an object to a specified type

```
Vector z = new Vector();
boolean b =
    z instanceof Vector;
// b== true;
```

Bitwise and Bit Shift Operators

(not that important for us; see URL below)

http://docs.oracle.com/javase/tutorial/java/nutsandbolts/op3.html



2 Language Basics – Operators

Equality and Relational Operators

==	Equal to	`boolean a = (1 == 1);`	`// a == true`
!=	Not equal to	`boolean a = (1 != 1);`	`// a == false`
>	Greater than	`boolean a = (17 > 12));`	`// a == true;`
>=	Greater than or equal to	etc.	
<	Less than		
<=	Less than or equal to		

Conditional Operators

&&	Conditional-AND	`a = false; b = true; c = a && b;`	`// c == false;`
||	Conditional-OR	`a = false; b = true; c = a || b;`	`// c == true;`
?:	Ternary (shorthand for if-then-else statement, use `if-then-else` instead!)		

Reference Type Comparison Operator

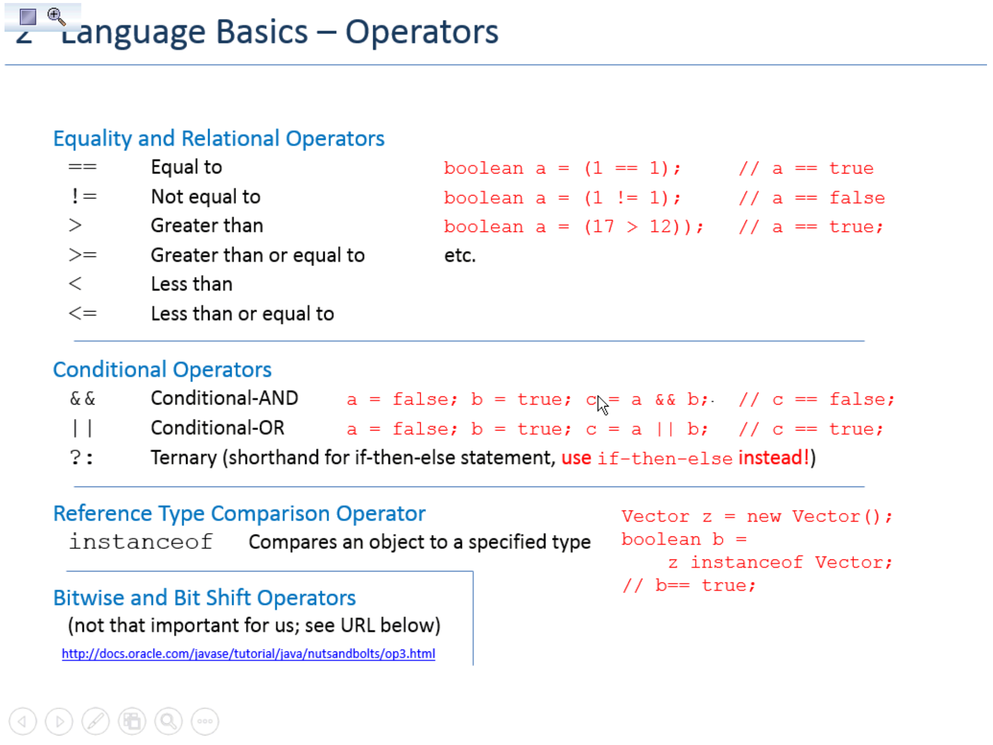
`instanceof` Compares an object to a specified type

```
Vector z = new Vector();
boolean b =
    z instanceof Vector;
// b== true;
```

Bitwise and Bit Shift Operators

(not that important for us; see URL below)

http://docs.oracle.com/javase/tutorial/java/nutsandbolts/op3.html



2 Language Basics – Operators

Equality and Relational Operators

==	Equal to	`boolean a = (1 == 1);`	`// a == true`
!=	Not equal to	`boolean a = (1 != 1);`	`// a == false`
>	Greater than	`boolean a = (17 > 12));`	`// a == true;`
>=	Greater than or equal to	etc.	
<	Less than		
<=	Less than or equal to		

Conditional Operators

&&	Conditional-AND	`a = false; b = true; c = a && b;`	`// c == false;`
||	Conditional-OR	`a = false; b = true; c = a || b;`	`// c == true;`
?:	Ternary (shorthand for if-then-else statement, use `if-then-else` instead!)		

Reference Type Comparison Operator

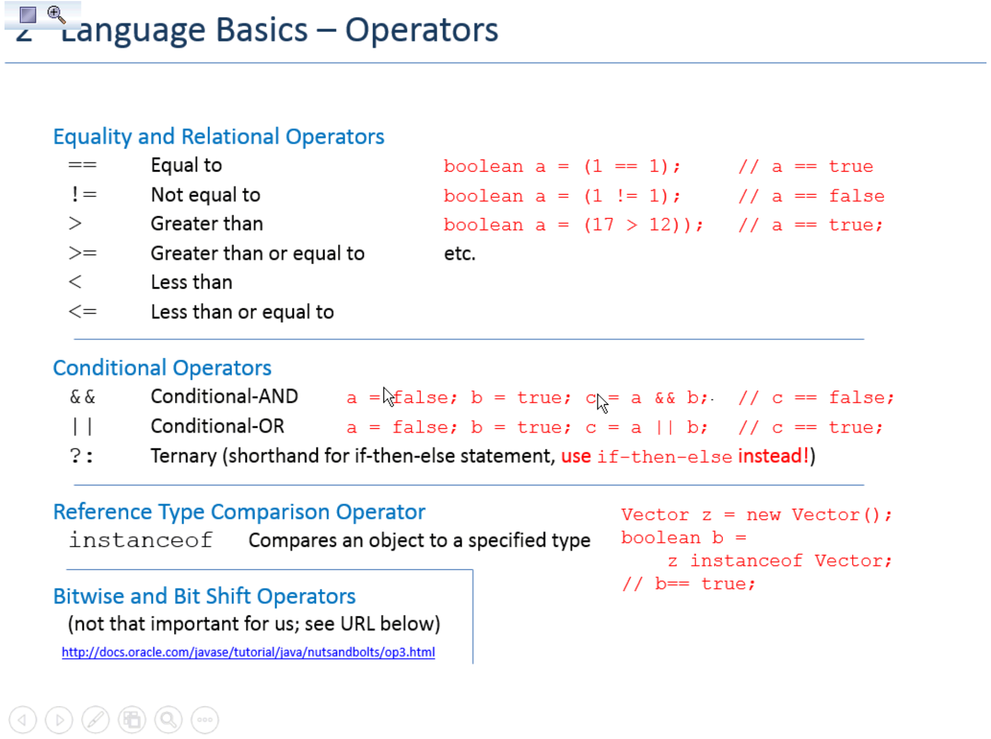
`instanceof` Compares an object to a specified type

```
Vector z = new Vector();
boolean b =
    z instanceof Vector;
// b== true;
```

Bitwise and Bit Shift Operators

(not that important for us; see URL below)

http://docs.oracle.com/javase/tutorial/java/nutsandbolts/op3.html



2 Language Basics – Operators

Equality and Relational Operators

==	Equal to	`boolean a = (1 == 1);`	`// a == true`
!=	Not equal to	`boolean a = (1 != 1);`	`// a == false`
>	Greater than	`boolean a = (17 > 12));`	`// a == true;`
>=	Greater than or equal to	etc.	
<	Less than		
<=	Less than or equal to		

Conditional Operators

&&	Conditional-AND	`a = false; b = true; c = a && b;`	`// c == false;`
||	Conditional-OR	`a = false; b = true; c = a || b;`	`// c == true;`
?:	Ternary (shorthand for if-then-else statement, use `if-then-else` instead!)		

Reference Type Comparison Operator

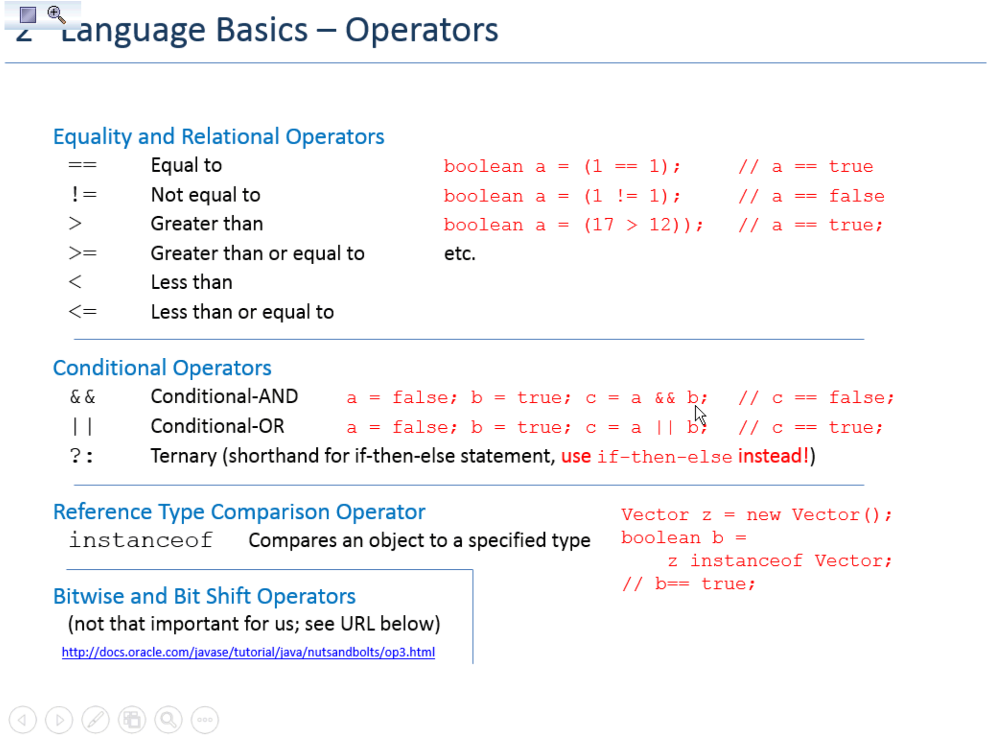
`instanceof` Compares an object to a specified type

```
Vector z = new Vector();
boolean b =
    z instanceof Vector;
// b== true;
```

Bitwise and Bit Shift Operators

(not that important for us; see URL below)

http://docs.oracle.com/javase/tutorial/java/nutsandbolts/op3.html



2 Language Basics – Operators

Equality and Relational Operators

==	Equal to	`boolean a = (1 == 1);`	`// a == true`
!=	Not equal to	`boolean a = (1 != 1);`	`// a == false`
>	Greater than	`boolean a = (17 > 12));`	`// a == true;`
>=	Greater than or equal to	etc.	
<	Less than		
<=	Less than or equal to		

Conditional Operators

&&	Conditional-AND	`a = false; b = true; c = a && b;`	`// c == false;`
||	Conditional-OR	`a = false; b = true; c = a || b;`	`// c == true;`
?:	Ternary (shorthand for if-then-else statement, use `if-then-else` instead!)		

Reference Type Comparison Operator

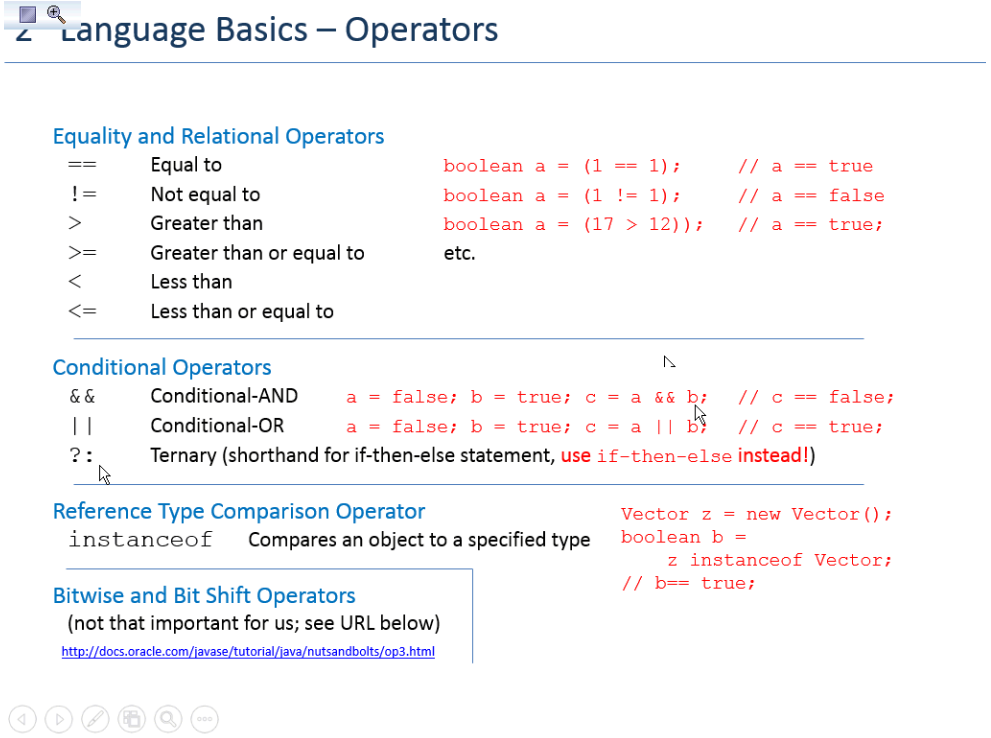
`instanceof` Compares an object to a specified type

```
Vector z = new Vector();
boolean b =
    z instanceof Vector;
// b== true;
```

Bitwise and Bit Shift Operators

(not that important for us; see URL below)

http://docs.oracle.com/javase/tutorial/java/nutsandbolts/op3.html



2 Language Basics – Operators

Equality and Relational Operators

==	Equal to	`boolean a = (1 == 1);`	`// a == true`
!=	Not equal to	`boolean a = (1 != 1);`	`// a == false`
>	Greater than	`boolean a = (17 > 12));`	`// a == true;`
>=	Greater than or equal to	etc.	
<	Less than		
<=	Less than or equal to		

Conditional Operators

&&	Conditional-AND	`a = false; b = true; c = a && b;`	`// c == false;`
||	Conditional-OR	`a = false; b = true; c = a || b;`	`// c == true;`
?:	Ternary (shorthand for if-then-else statement, use `if-then-else` instead!)		

Reference Type Comparison Operator

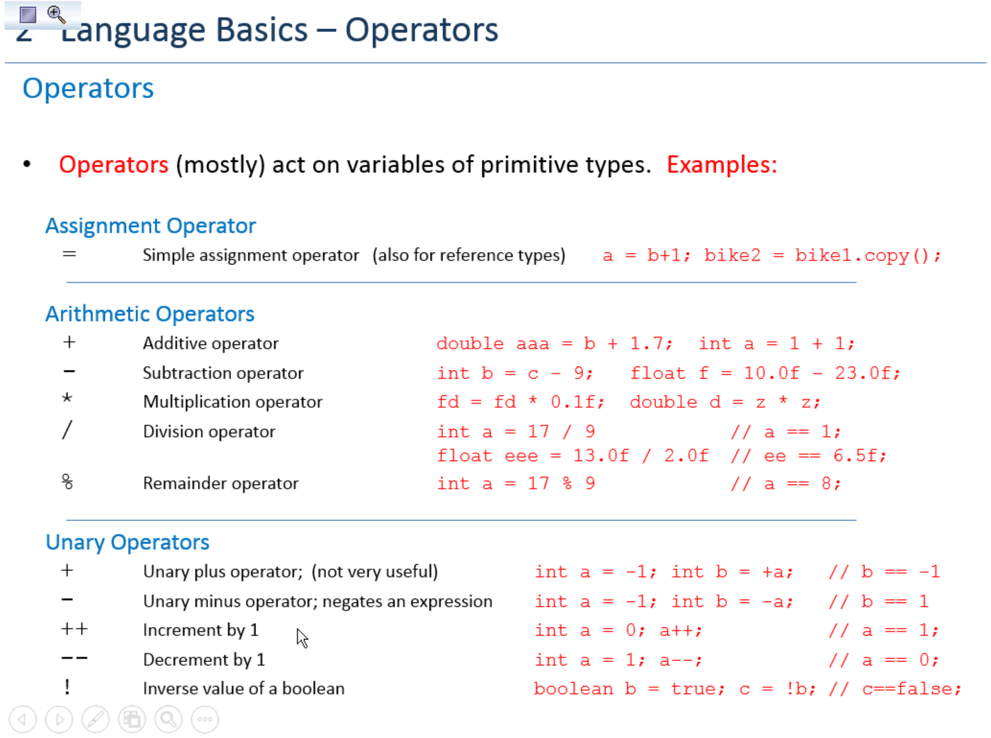
`instanceof` Compares an object to a specified type

```
Vector z = new Vector();
boolean b =
    z instanceof Vector;
// b== true;
```

Bitwise and Bit Shift Operators

(not that important for us; see URL below)

http://docs.oracle.com/javase/tutorial/java/nutsandbolts/op3.html



2 Language Basics – Operators

Operators

- Operators (mostly) act on variables of primitive types. Examples:

Assignment Operator

=	Simple assignment operator (also for reference types)	`a = b+1; bike2 = bike1.copy();`

Arithmetic Operators

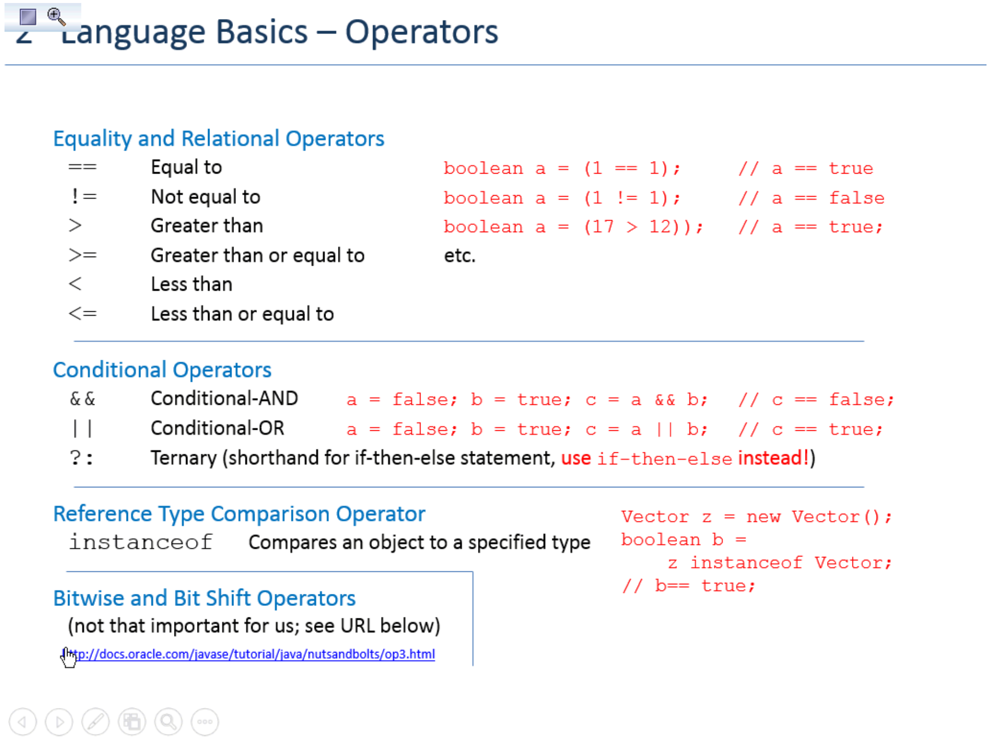
+	Additive operator	`double aaa = b + 1.7;  int a = 1 + 1;`
-	Subtraction operator	`int b = c - 9;  float f = 10.0f - 23.0f;`
*	Multiplication operator	`fd = fd * 0.1f;  double d = z * z;`

/	Division operator	`int a = 17 / 9          // a == 1;`
		`float eee = 13.0f / 2.0f  // ee == 6.5f;`

%	Remainder operator	`int a = 17 % 9          // a == 8;`

Unary Operators

+	Unary plus operator; (not very useful)	`int a = -1; int b = +a;`	`// b == -1`
-	Unary minus operator; negates an expression	`int a = -1; int b = -a;`	`// b == 1`
++	Increment by 1	`int a = 0; a++;`	`// a == 1;`
--	Decrement by 1	`int a = 1; a--;`	`// a == 0;`
!	Inverse value of a boolean	`boolean b = true; c = !b;`	`// c==false;`



2 Language Basics – Operators

Equality and Relational Operators

==	Equal to	`boolean a = (1 == 1);      // a == true`
!=	Not equal to	`boolean a = (1 != 1);      // a == false`
>	Greater than	`boolean a = (17 > 12));    // a == true;`
>=	Greater than or equal to	etc.
<	Less than	
<=	Less than or equal to	

Conditional Operators

&&	Conditional-AND	`a = false; b = true; c = a && b;   // c == false;`
||	Conditional-OR	`a = false; b = true; c = a || b;   // c == true;`
?:	Ternary (shorthand for if-then-else statement, use `if-then-else` instead!)	

Reference Type Comparison Operator

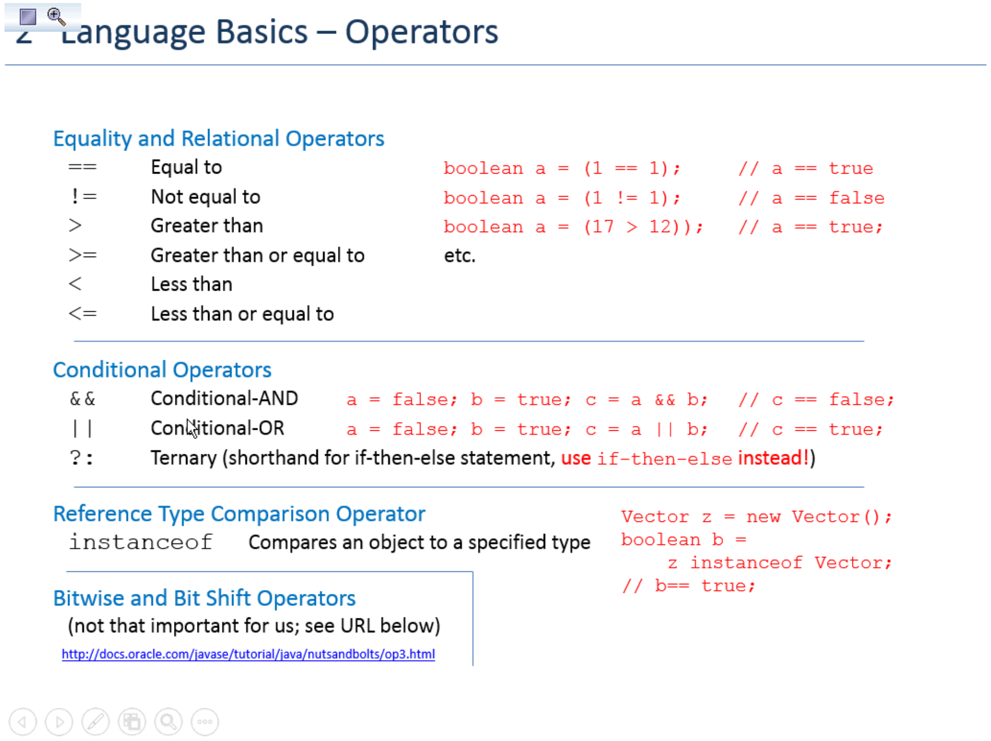
`instanceof` Compares an object to a specified type

```
Vector z = new Vector();
boolean b =
    z instanceof Vector;
// b== true;
```

Bitwise and Bit Shift Operators

(not that important for us; see URL below)

http://docs.oracle.com/javase/tutorial/java/nutsandbolts/op3.html



2 Language Basics – Operators

Equality and Relational Operators

==	Equal to	`boolean a = (1 == 1);      // a == true`
!=	Not equal to	`boolean a = (1 != 1);      // a == false`
>	Greater than	`boolean a = (17 > 12));    // a == true;`
>=	Greater than or equal to	etc.
<	Less than	
<=	Less than or equal to	

Conditional Operators

&&	Conditional-AND	`a = false; b = true; c = a && b;   // c == false;`
||	Conditional-OR	`a = false; b = true; c = a || b;   // c == true;`
?:	Ternary (shorthand for if-then-else statement, use `if-then-else` instead!)	

Reference Type Comparison Operator

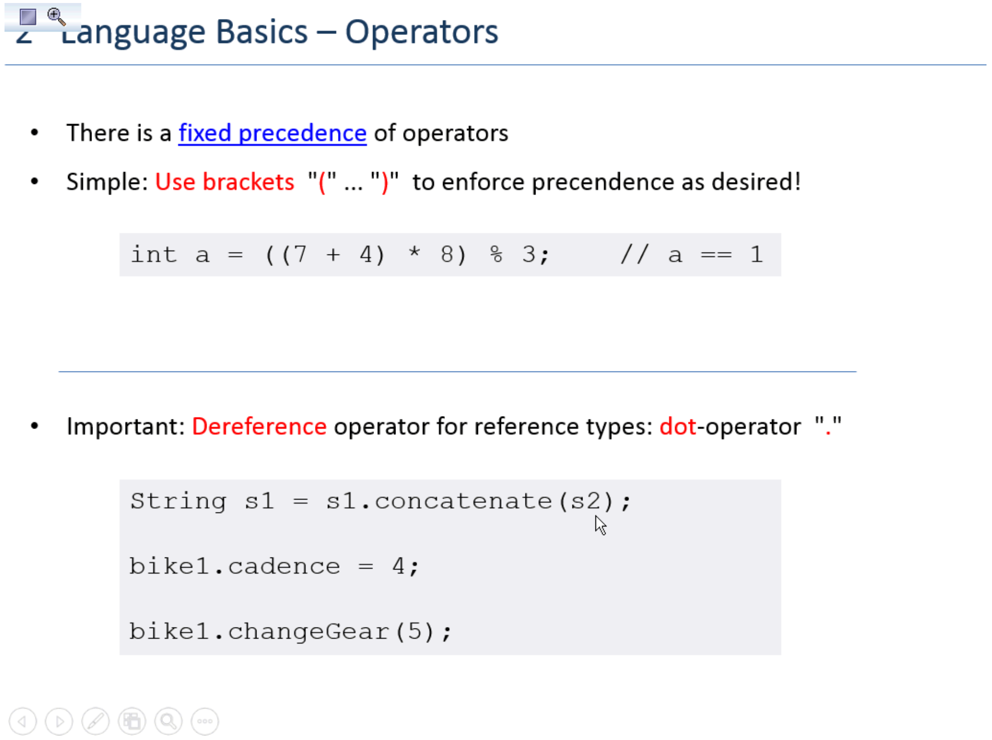
`instanceof` Compares an object to a specified type

```
Vector z = new Vector();
boolean b =
    z instanceof Vector;
// b== true;
```

Bitwise and Bit Shift Operators

(not that important for us; see URL below)

http://docs.oracle.com/javase/tutorial/java/nutsandbolts/op3.html



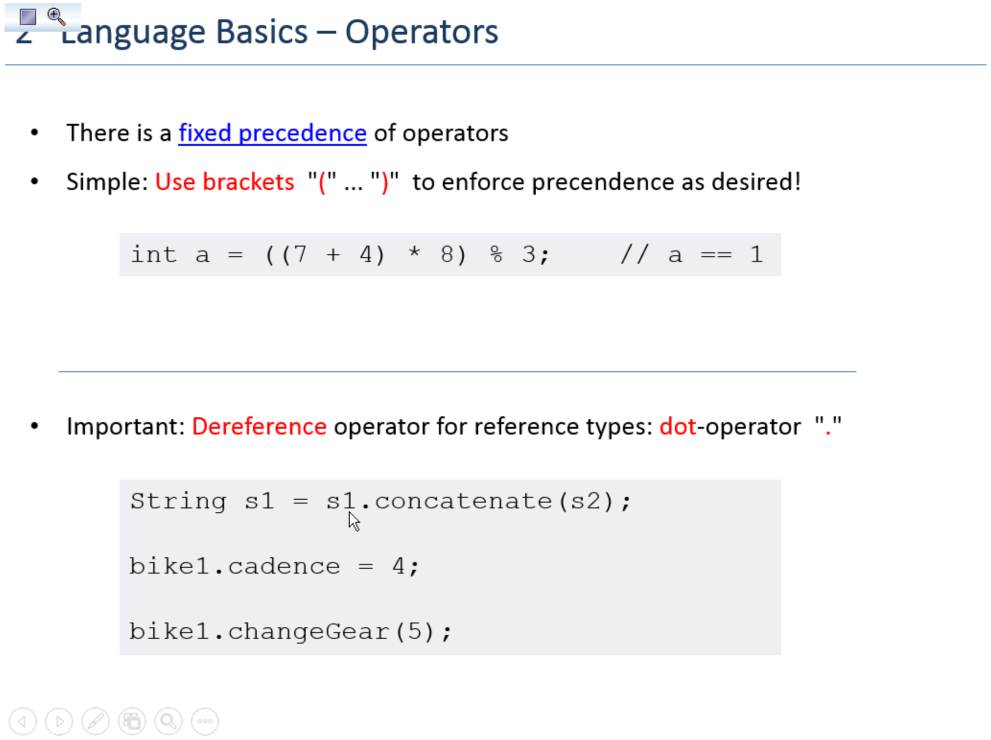
2 Language Basics – Operators

- There is a fixed precedence of operators
- Simple: Use brackets "(" ... ")" to enforce precendence as desired!

```
int a = ((7 + 4) * 8) % 3;     // a == 1
```

- Important: Dereference operator for reference types: dot-operator "."

```
String s1 = s1.concatenate(s2);

bike1.cadence = 4;

bike1.changeGear(5);
```



2 Language Basics – Operators

- There is a fixed precedence of operators
- Simple: Use brackets "(" ... ")" to enforce precendence as desired!

```
int a = ((7 + 4) * 8) % 3;     // a == 1
```

- Important: Dereference operator for reference types: dot-operator "."

```
String s1 = s1.concatenate(s2);

bike1.cadence = 4;

bike1.changeGear(5);
```

2 Language Basics – Operators

- There is a fixed precedence of operators
- Simple: Use brackets "(" ... ")" to enforce precendence as desired!

```
int a = ((7 + 4) * 8) % 3;    // a == 1
```

- Important: Dereference operator for reference types: dot-operator "."

```
String s1 = s1.concatenate(s2);

bike1.cadence = 4;

bike1.changeGear(5);
```

2 Language Basics – Operators

- There is a fixed precedence of operators
- Simple: Use brackets "(" ... ")" to enforce precendence as desired!

```
int a = ((7 + 4) * 8) % 3;    // a == 1
```

- Important: Dereference operator for reference types: dot-operator "."

```
String s1 = s1.concatenate(s2);

bike1.cadence = 4;

bike1.changeGear(5);
```

2 Language Basics – Operators

- There is a fixed precedence of operators
- Simple: Use brackets "(" ... ")" to enforce precendence as desired!

```
int a = ((7 + 4) * 8) % 3;    // a == 1
```

- Important: Dereference operator for reference types: dot-operator "."

```
String s1 = s1.concatenate(s2);

bike1.cadence = 4;

bike1.changeGear(5);
```

2 Language Basics – Expressions, Statements, Blocks

Expressions

- Some expressions have so-called side-effects (in most cases the only important aspect about the expression!!!)

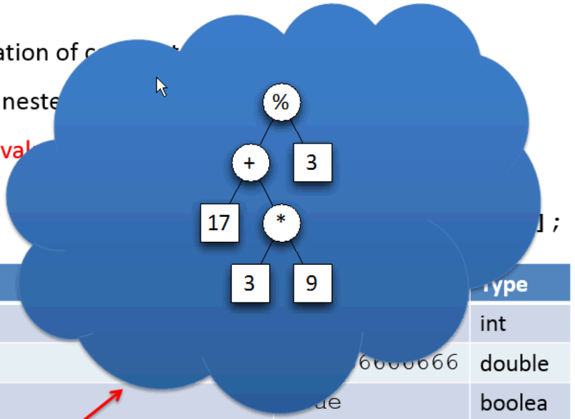
Given: **`int a = 73;   int b;`**

Example	Value	Side-effect
`a = 84`	84	Assign 84 to `a`
`b = (a = 48)`	48	Assign 48 to both `a` and `b`
`a++`	48	Assign 49 to `a` (!)
`++a`	50	Assign 50 to `a` (!)
`new Bicycle()`	Reference to the new instance of `Bicycle`, e.g. `<1150>`	Create and initialize new instance of class `Bicycle` in memory
`new double[10]`	Reference to the new array of `double`	Create and initialize new array in memory

2 Language Basics – Expressions, Statements, Blocks

Expressions

- Expression: Legal combination of c
- Can be (and typically are) neste
- Expressions evaluate to a val



Given: `int a = 73;`] ;

Example		Type
`48`		int
`2.0 / 3.0`	`6666666`	double
`true`	`ue`	boolean
`15 / 8`	`1`	int
`(17 + (3 * 9)) % 3`	`2`	int
`a + 1`	`74`	int
`a * 9.0 / someArray.length`	`131.4`	double

2 Language Basics – Expressions, Statements, Blocks

Expressions

- Some expressions have so-called side-effects (in most cases the only important aspect about the expression!!!)

Given: `int a = 73; int b;`

Example	Value	Side-effect
`a = 84`	`84`	Assign 84 to `a`
`b = (a = 48)`	`48`	Assign 48 to both `a` and `b`
`a++`	`48`	Assign 49 to `a` (!)
`++a`	`50`	Assign 50 to `a` (!)
`new Bicycle()`	Reference to the new instance of `Bicycle`, e.g. `<1150>`	Create and initialize new instance of class `Bicycle` in memory
`new double[10]`	Reference to the new array of `double`	Create and initialize new array in memory

2 Language Basics – Expressions, Statements, Blocks

Expressions

- Some expressions have so-called side-effects (in most cases the only important aspect about the expression!!!)

Given: `int a = 73; int b;`

Example	Value	Side-effect
`a = 84`	`84`	Assign 84 to `a`
`b = (a = 48)`	`48`	Assign 48 to both `a` and `b`
`a++`	`48`	Assign 49 to `a` (!)
`++a`	`50`	Assign 50 to `a` (!)
`new Bicycle()`	Reference to the new instance of `Bicycle`, e.g. `<1150>`	Create and initialize new instance of class `Bicycle` in memory
`new double[10]`	Reference to the new array of `double`	Create and initialize new array in memory

2 Language Basics – Expressions, Statements, Blocks

Expressions

- Some expressions have so-called side-effects (in most cases the only important aspect about the expression!!!)

Given: `int a = 73; int b;`

Example	Value	Side-effect
`a = 84`	`84`	Assign 84 to `a`
`b = (a = 48)`	`48`	Assign 48 to both `a` and `b`
`a++`	`48`	Assign 49 to `a` (!)
`++a`	`50`	Assign 50 to `a` (!)
`new Bicycle()`	Reference to the new instance of `Bicycle`, e.g. `<1150>`	Create and initialize new instance of class `Bicycle` in memory
`new double[10]`	Reference to the new array of `double`	Create and initialize new array in memory

2 Language Basics – Expressions, Statements, Blocks

Statements

- Statement: Complete unit of execution (ends with ";")
  - Expression statements:

Assigment expressions	`a = (17 + (3 * 9)) % 3;`
Use of ++ or --	`a++;`
Method invocations	`someObject.methodOne();`
Object creation expressions	`new SomeClass();`
Declaration statements	`int a;`

  - Blocks
    - (next slide)
  - Control flow statements
    - (later)

2 Language Basics – Expressions, Statements, Blocks

Blocks

- Variables declared inside a block are only visible from within that block:

```
int a = 7, b = 6;

if (a != b) {              // begin block
  int c;
  c = a * b;
  System.out.println(c);
}                          // end block

System.out.println(c);     // ERROR: c
unavailable
```

2 Language Basics – Expressions, Statements, Blocks

Blocks

- Variables declared inside a block are only visible from within that block:

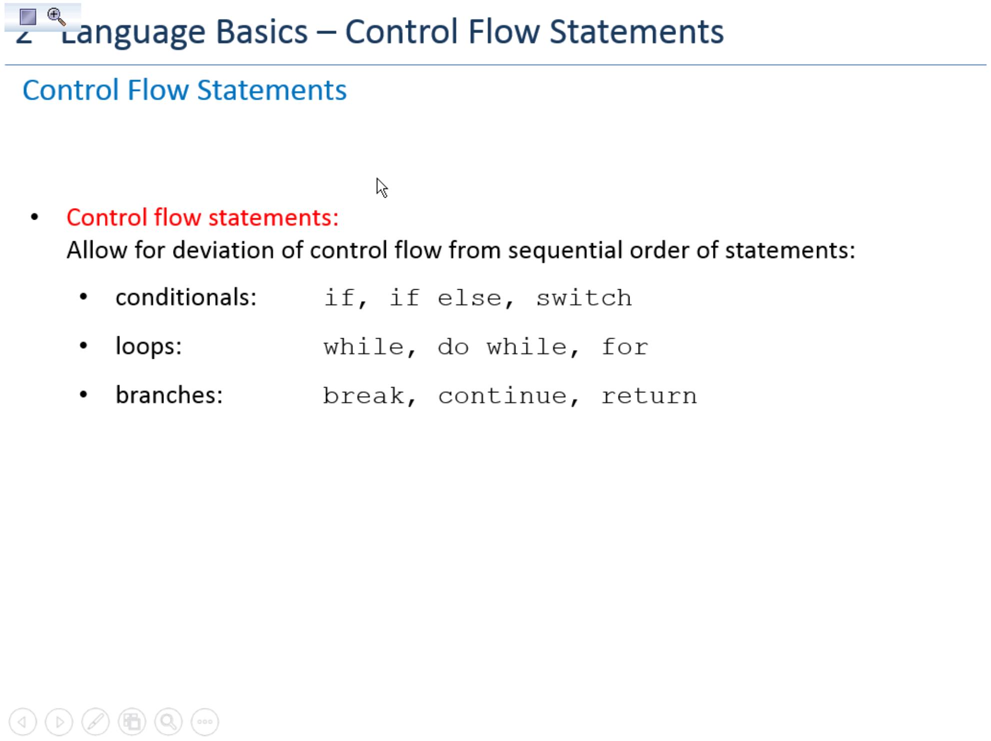
```
int a = 7, b = 6;

if (a != b) {              // begin block
  int c;
  c = a * b;
  System.out.println(c);
}                          // end block

System.out.println(c);     // ERROR: c
unavailable
```

2 Language Basics – Control Flow Statements

Control Flow Statements

- Control flow statements:
  Allow for deviation of control flow from sequential order of statements:

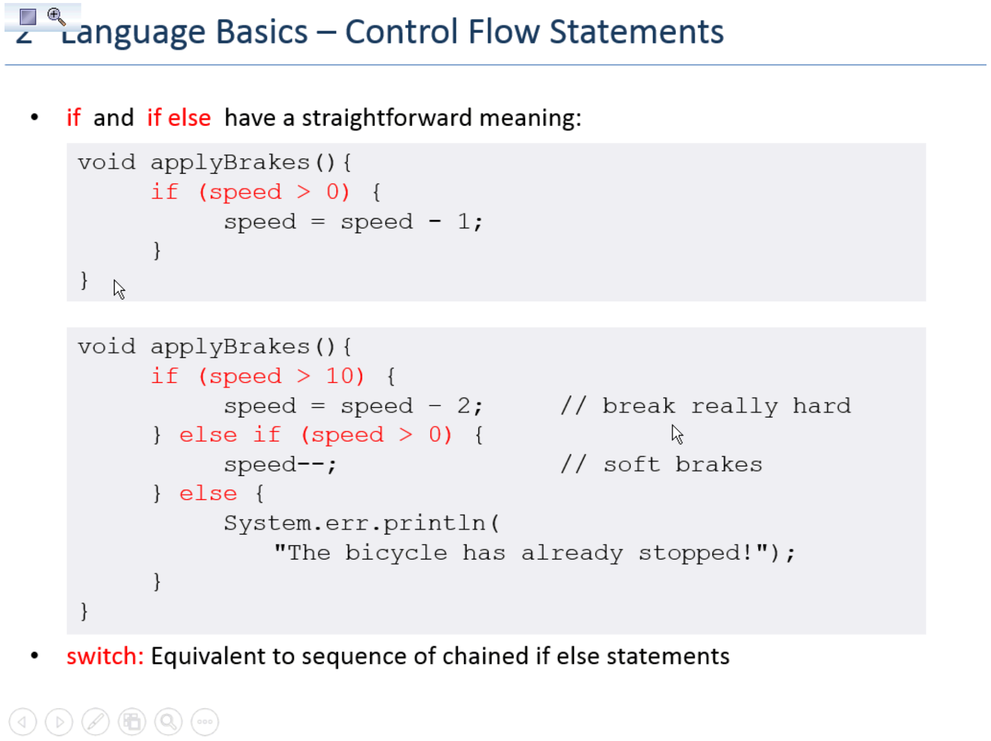
conditionals:	`if, if else, switch`
loops:	`while, do while, for`
branches:	`break, continue, return`



2 Language Basics – Control Flow Statements

Control Flow Statements

- Control flow statements:
  Allow for deviation of control flow from sequential order of statements:
  - conditionals: `if, if else, switch`
  - loops: `while, do while, for`
  - branches: `break, continue, return`



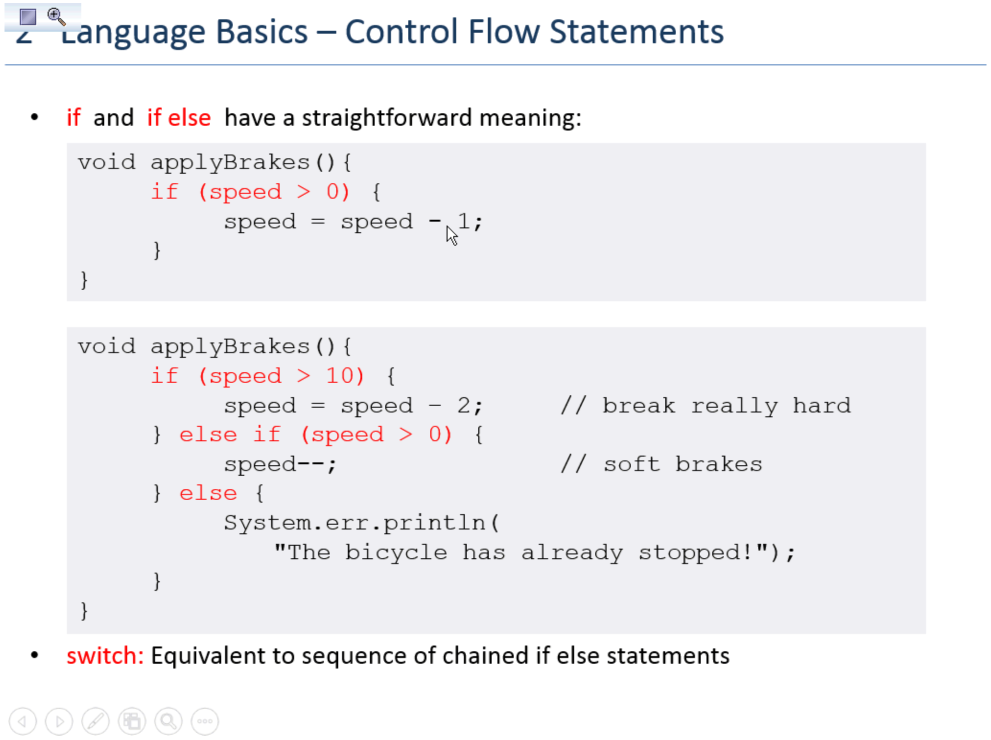
2 Language Basics – Control Flow Statements

- if and if else have a straightforward meaning:

```
void applyBrakes(){
     if (speed > 0) {
          speed = speed - 1;
     }
}
```

```
void applyBrakes(){
     if (speed > 10) {
          speed = speed - 2;      // break really hard
     } else if (speed > 0) {
          speed--;                // soft brakes
     } else {
          System.err.println(
              "The bicycle has already stopped!");
     }
}
```

- switch: Equivalent to sequence of chained if else statements



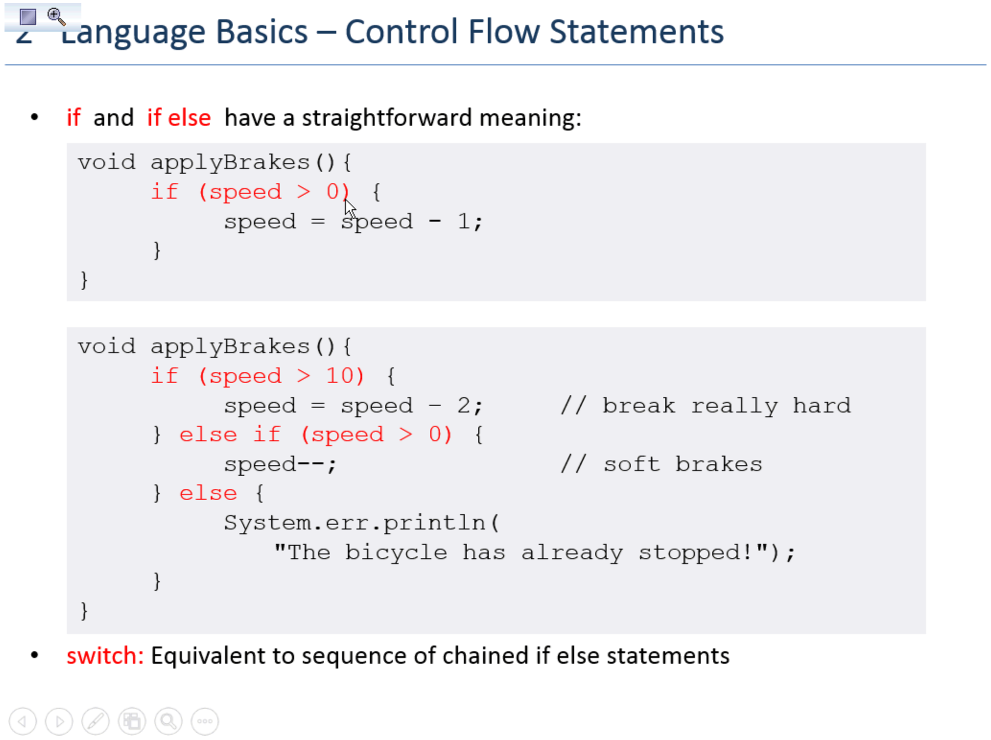
2 Language Basics – Control Flow Statements

- if and if else have a straightforward meaning:

```
void applyBrakes(){
     if (speed > 0) {
          speed = speed - 1;
     }
}
```

```
void applyBrakes(){
     if (speed > 10) {
          speed = speed - 2;      // break really hard
     } else if (speed > 0) {
          speed--;                // soft brakes
     } else {
          System.err.println(
              "The bicycle has already stopped!");
     }
}
```

- switch: Equivalent to sequence of chained if else statements



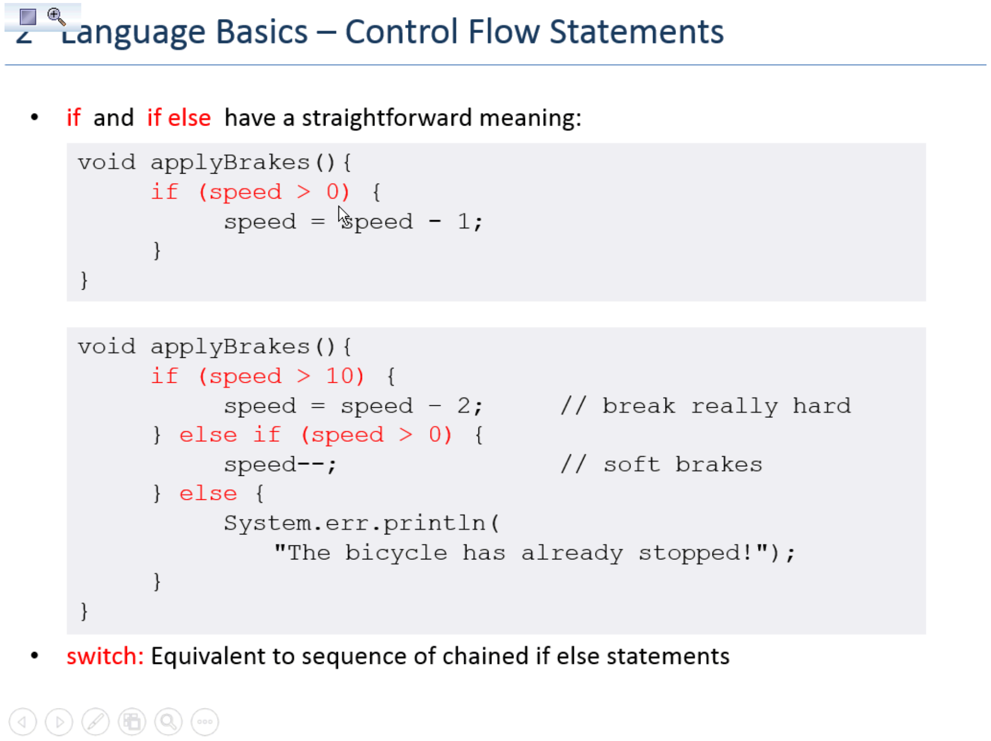
2 Language Basics – Control Flow Statements

- if and if else have a straightforward meaning:

```
void applyBrakes(){
     if (speed > 0) {
          speed = speed - 1;
     }
}
```

```
void applyBrakes(){
     if (speed > 10) {
          speed = speed - 2;      // break really hard
     } else if (speed > 0) {
          speed--;                // soft brakes
     } else {
          System.err.println(
              "The bicycle has already stopped!");
     }
}
```

- switch: Equivalent to sequence of chained if else statements



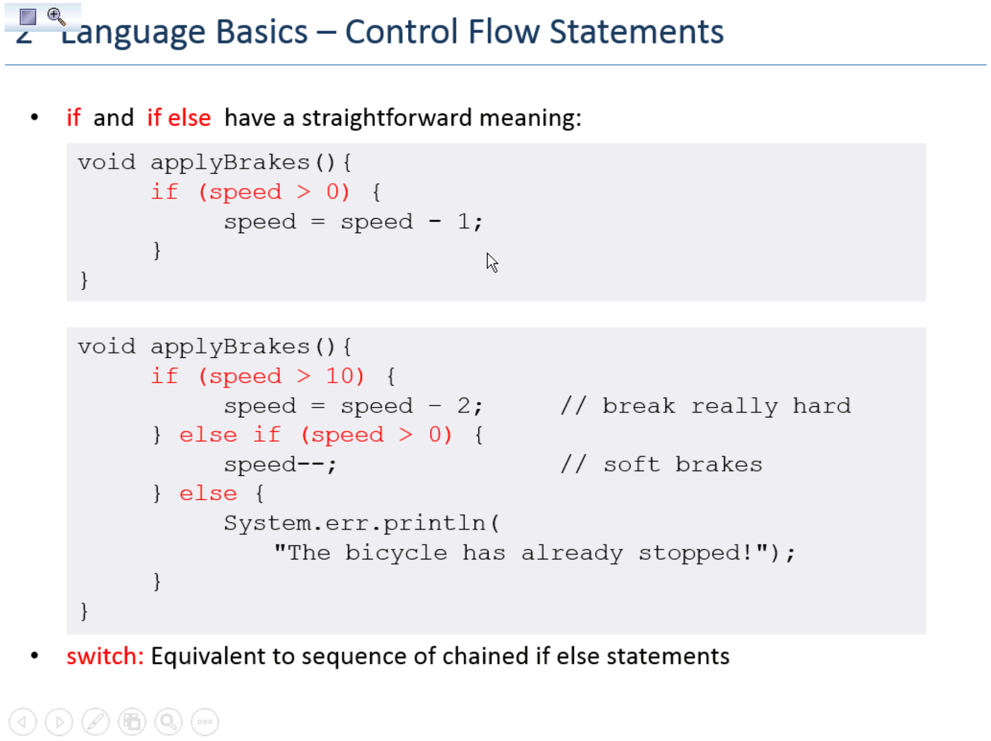
2 Language Basics – Control Flow Statements

- if and if else have a straightforward meaning:

```
void applyBrakes(){
     if (speed > 0) {
          speed = speed - 1;
     }
}
```

```
void applyBrakes(){
     if (speed > 10) {
          speed = speed - 2;       // break really hard
     } else if (speed > 0) {
          speed--;                 // soft brakes
     } else {
          System.err.println(
              "The bicycle has already stopped!");
     }
}
```

- switch: Equivalent to sequence of chained if else statements



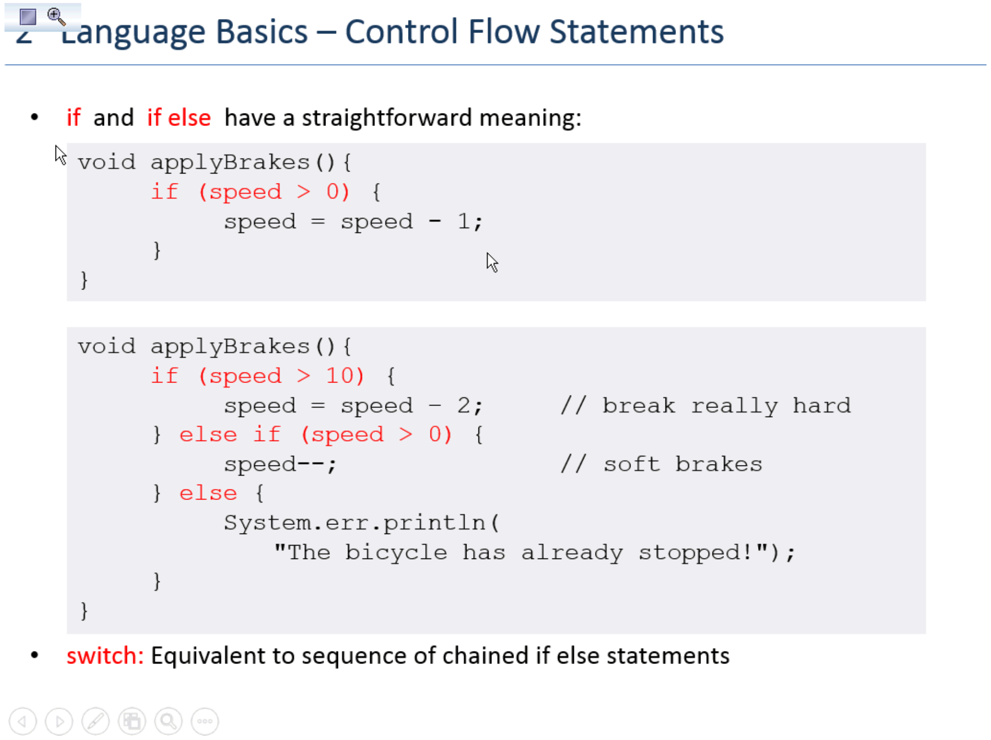
2 Language Basics – Control Flow Statements

- if and if else have a straightforward meaning:

```
void applyBrakes(){
     if (speed > 0) {
          speed = speed - 1;
     }
}
```

```
void applyBrakes(){
     if (speed > 10) {
          speed = speed - 2;       // break really hard
     } else if (speed > 0) {
          speed--;                 // soft brakes
     } else {
          System.err.println(
              "The bicycle has already stopped!");
     }
}
```

- switch: Equivalent to sequence of chained if else statements



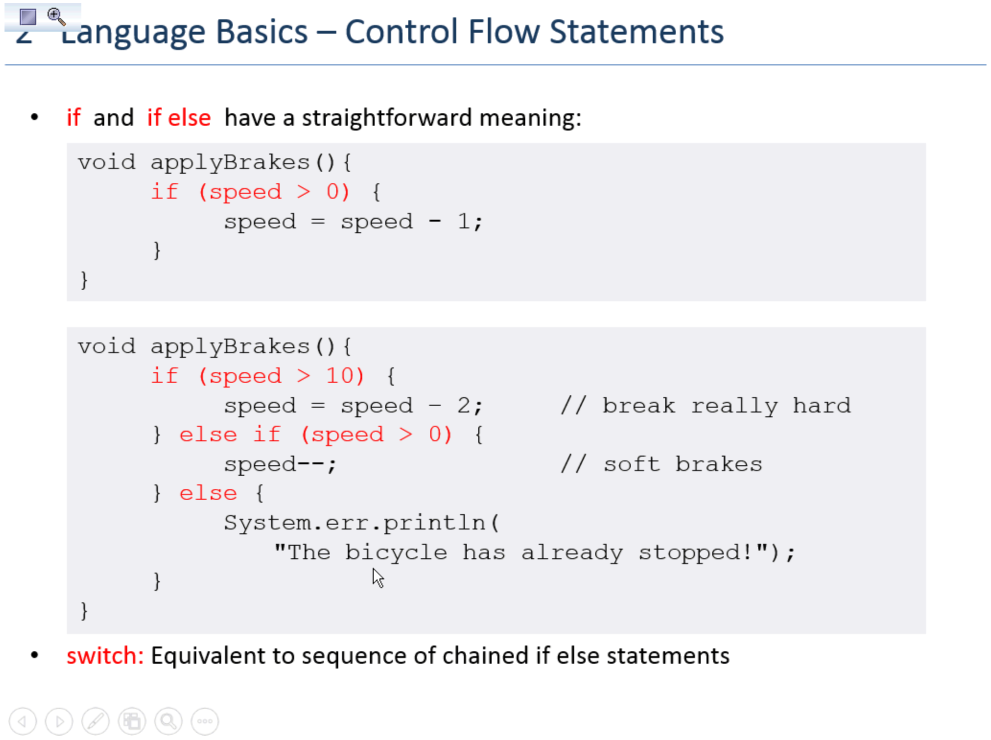
2 Language Basics – Control Flow Statements

- if and if else have a straightforward meaning:

```
void applyBrakes(){
     if (speed > 0) {
          speed = speed - 1;
     }
}
```

```
void applyBrakes(){
     if (speed > 10) {
          speed = speed - 2;       // break really hard
     } else if (speed > 0) {
          speed--;                 // soft brakes
     } else {
          System.err.println(
              "The bicycle has already stopped!");
     }
}
```

- switch: Equivalent to sequence of chained if else statements



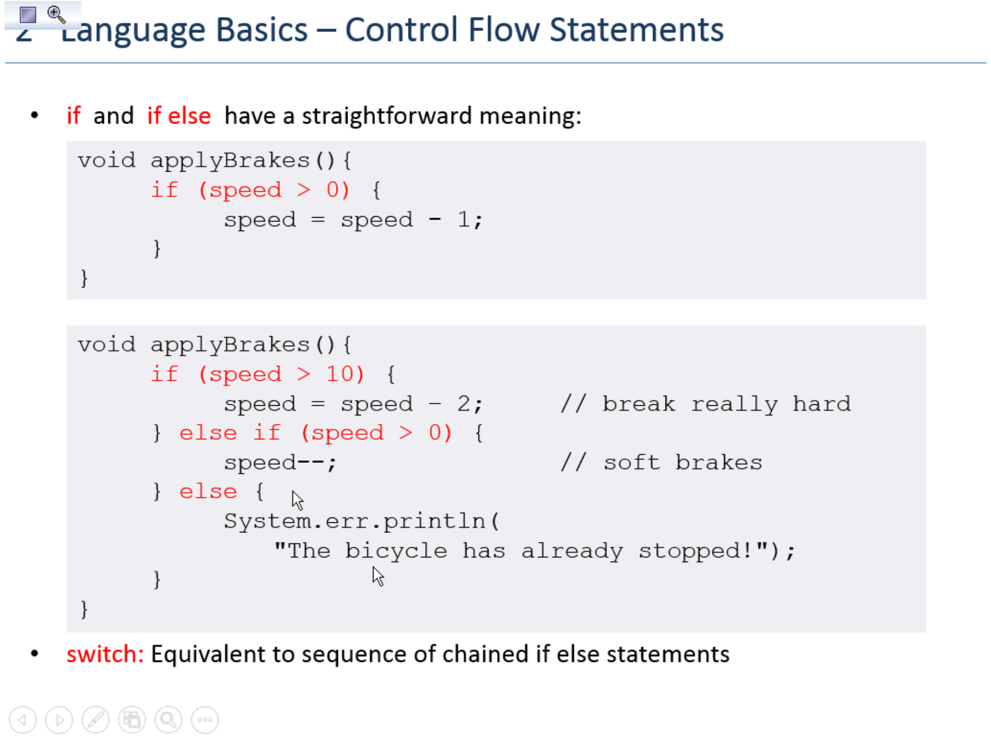
2 Language Basics – Control Flow Statements

- if and if else have a straightforward meaning:

```
void applyBrakes(){
     if (speed > 0) {
          speed = speed - 1;
     }
}
```

```
void applyBrakes(){
     if (speed > 10) {
          speed = speed - 2;       // break really hard
     } else if (speed > 0) {
          speed--;                 // soft brakes
     } else {
          System.err.println(
              "The bicycle has already stopped!");
     }
}
```

- switch: Equivalent to sequence of chained if else statements



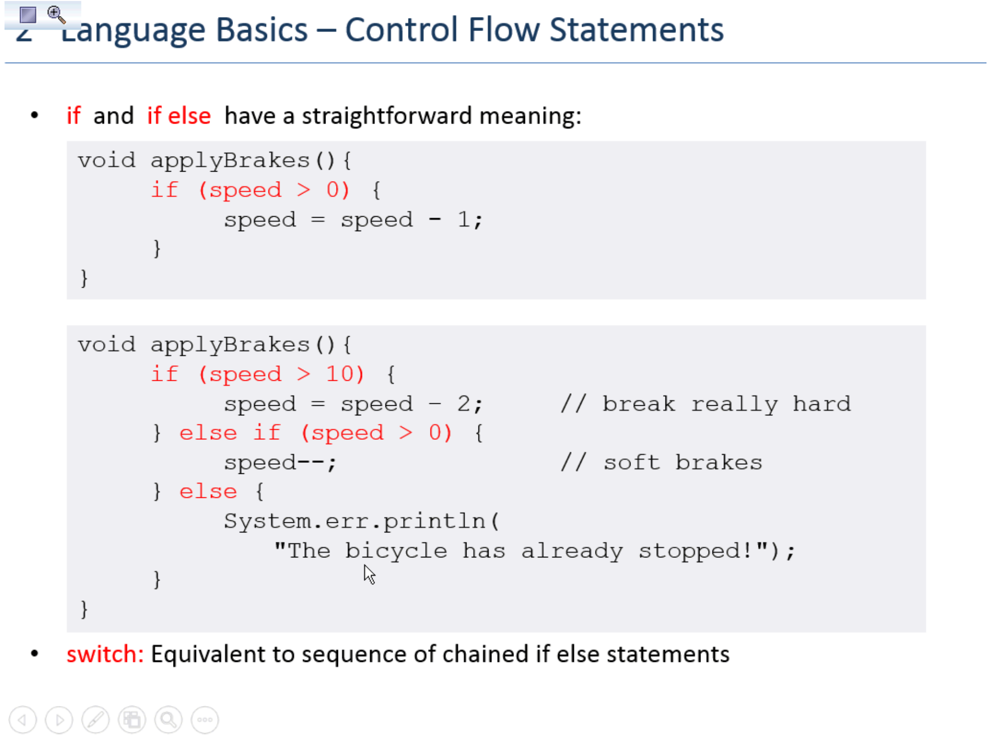
2 Language Basics – Control Flow Statements

- if and if else have a straightforward meaning:

```
void applyBrakes(){
     if (speed > 0) {
          speed = speed - 1;
     }
}
```

```
void applyBrakes(){
     if (speed > 10) {
          speed = speed - 2;      // break really hard
     } else if (speed > 0) {
          speed--;                // soft brakes
     } else {
          System.err.println(
             "The bicycle has already stopped!");
     }
}
```

- switch: Equivalent to sequence of chained if else statements



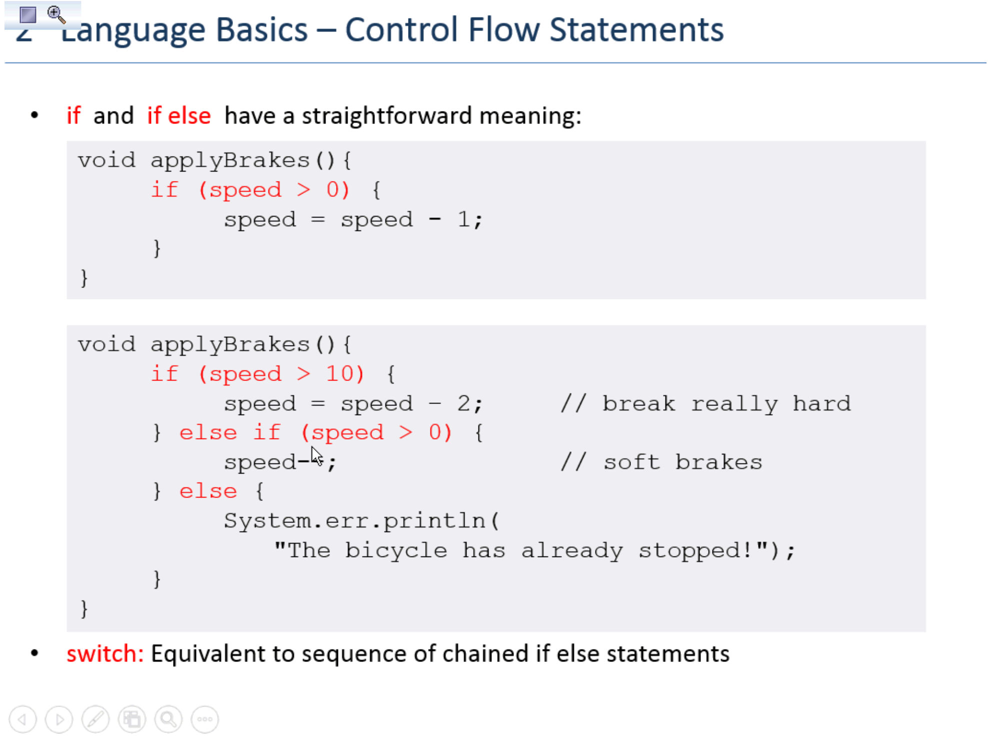
2 Language Basics – Control Flow Statements

- if and if else have a straightforward meaning:

```
void applyBrakes(){
     if (speed > 0) {
          speed = speed - 1;
     }
}
```

```
void applyBrakes(){
     if (speed > 10) {
          speed = speed - 2;      // break really hard
     } else if (speed > 0) {
          speed--;                // soft brakes
     } else {
          System.err.println(
             "The bicycle has already stopped!");
     }
}
```

- switch: Equivalent to sequence of chained if else statements



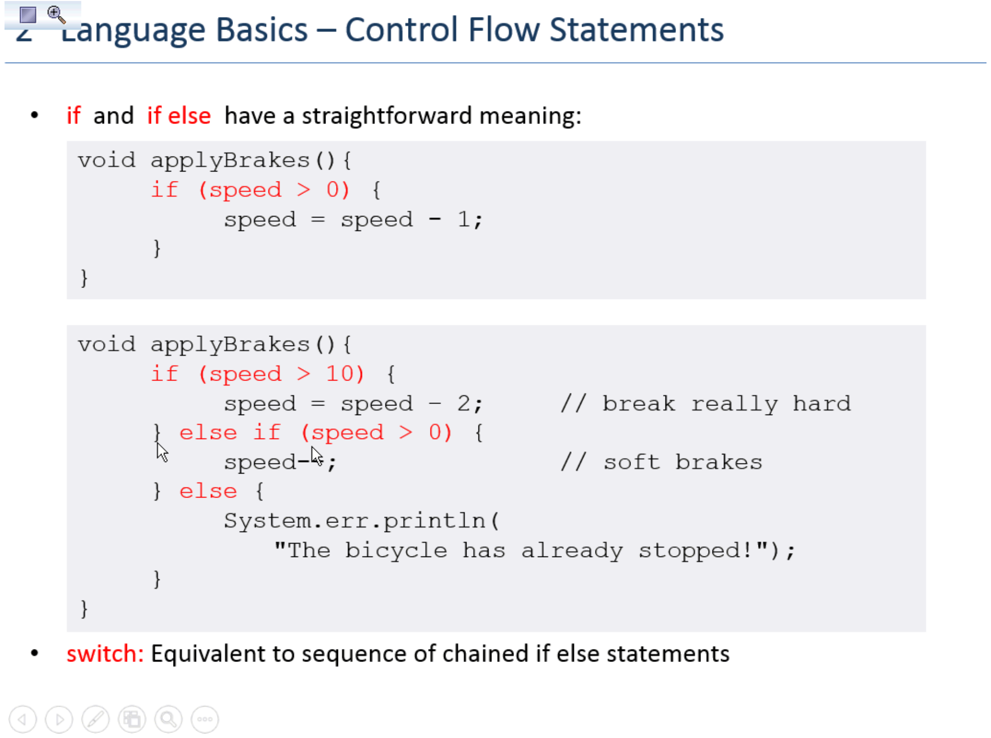
2 Language Basics – Control Flow Statements

- if and if else have a straightforward meaning:

```
void applyBrakes(){
     if (speed > 0) {
          speed = speed - 1;
     }
}
```

```
void applyBrakes(){
     if (speed > 10) {
          speed = speed - 2;      // break really hard
     } else if (speed > 0) {
          speed--;                // soft brakes
     } else {
          System.err.println(
             "The bicycle has already stopped!");
     }
}
```

- switch: Equivalent to sequence of chained if else statements



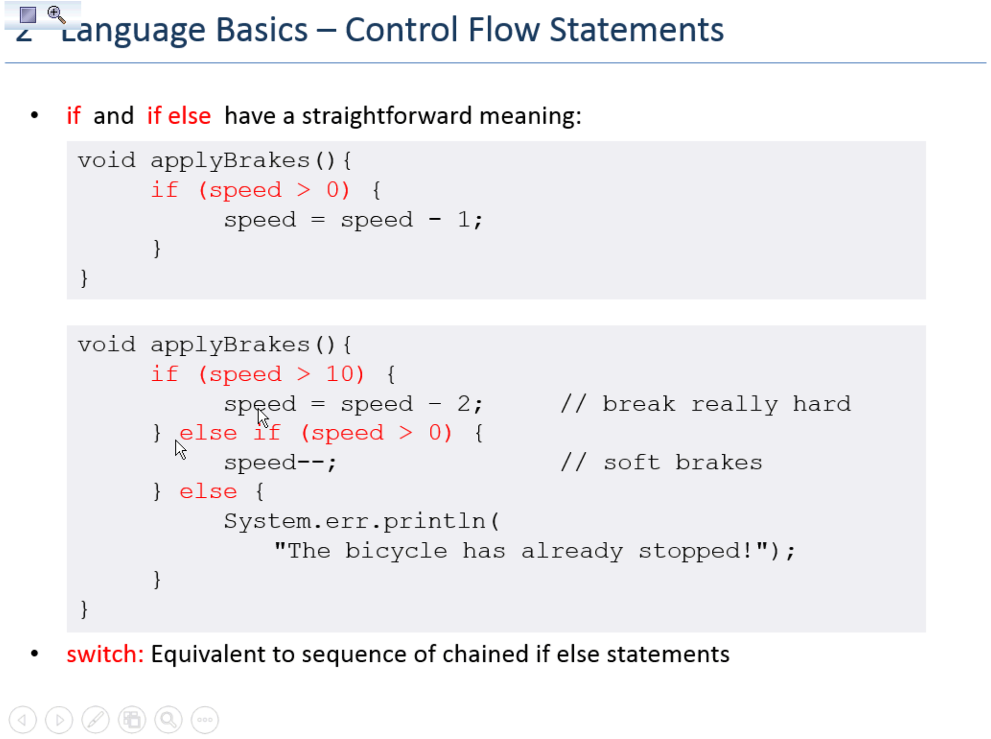
2 Language Basics – Control Flow Statements

- if and if else have a straightforward meaning:

```
void applyBrakes(){
     if (speed > 0) {
          speed = speed - 1;
     }
}
```

```
void applyBrakes(){
     if (speed > 10) {
          speed = speed - 2;      // break really hard
     } else if (speed > 0) {
          speed--;                // soft brakes
     } else {
          System.err.println(
             "The bicycle has already stopped!");
     }
}
```

- switch: Equivalent to sequence of chained if else statements



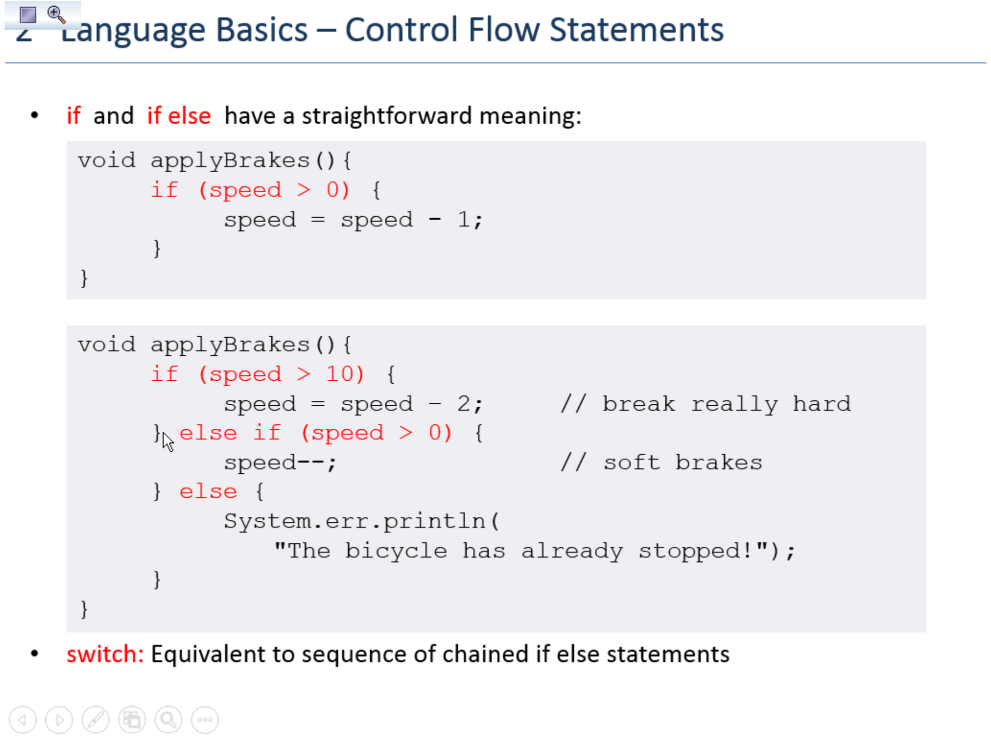
2 Language Basics – Control Flow Statements

- if and if else have a straightforward meaning:

```
void applyBrakes(){
     if (speed > 0) {
          speed = speed - 1;
     }
}
```

```
void applyBrakes(){
     if (speed > 10) {
          speed = speed - 2;      // break really hard
     } else if (speed > 0) {
          speed--;                // soft brakes
     } else {
          System.err.println(
              "The bicycle has already stopped!");
     }
}
```

- switch: Equivalent to sequence of chained if else statements



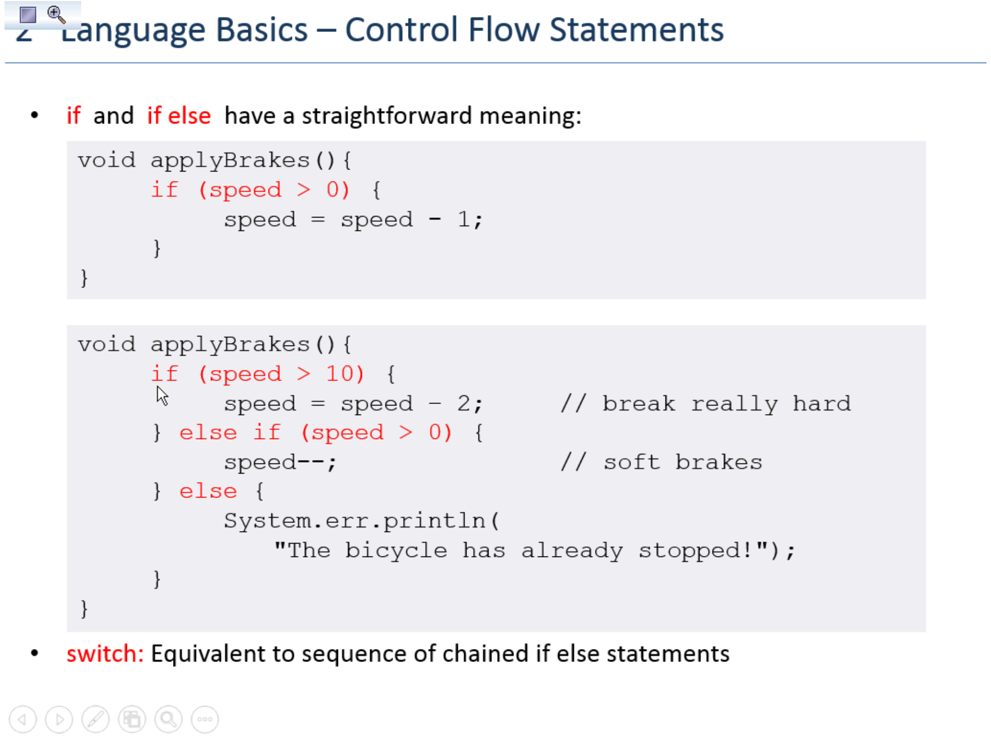
2 Language Basics – Control Flow Statements

- if and if else have a straightforward meaning:

```
void applyBrakes(){
     if (speed > 0) {
          speed = speed - 1;
     }
}
```

```
void applyBrakes(){
     if (speed > 10) {
          speed = speed - 2;      // break really hard
     } else if (speed > 0) {
          speed--;                // soft brakes
     } else {
          System.err.println(
              "The bicycle has already stopped!");
     }
}
```

- switch: Equivalent to sequence of chained if else statements



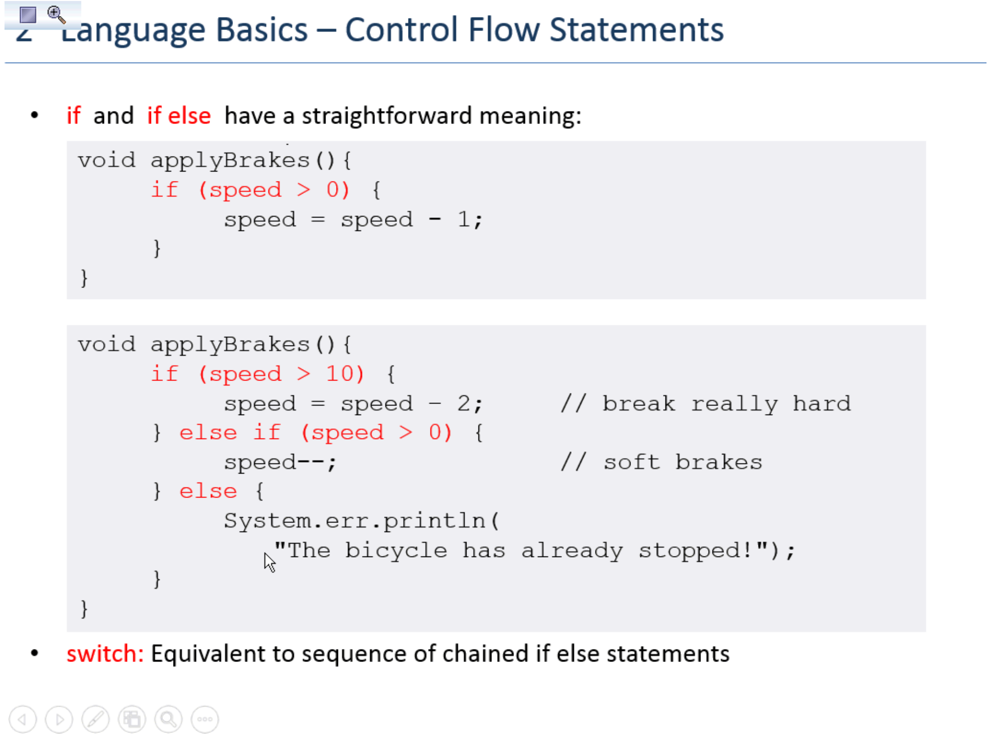
2 Language Basics – Control Flow Statements

- if and if else have a straightforward meaning:

```
void applyBrakes(){
     if (speed > 0) {
          speed = speed - 1;
     }
}
```

```
void applyBrakes(){
     if (speed > 10) {
          speed = speed - 2;      // break really hard
     } else if (speed > 0) {
          speed--;                // soft brakes
     } else {
          System.err.println(
              "The bicycle has already stopped!");
     }
}
```

- switch: Equivalent to sequence of chained if else statements



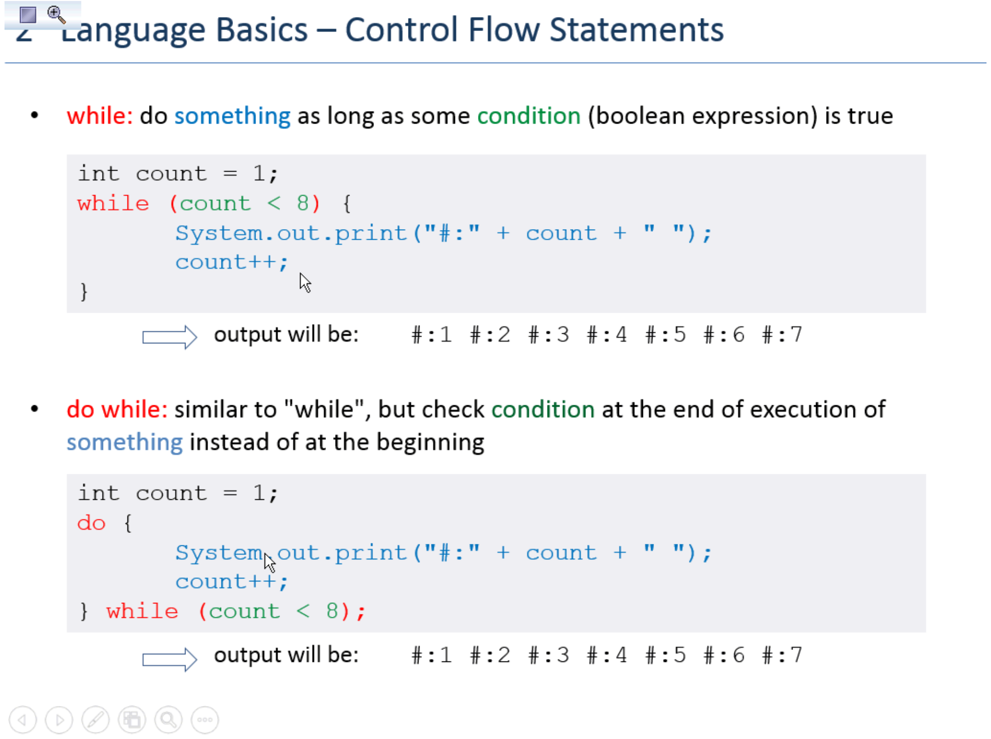
2 Language Basics – Control Flow Statements

- if and if else have a straightforward meaning:

```
void applyBrakes(){
     if (speed > 0) {
          speed = speed - 1;
     }
}
```

```
void applyBrakes(){
     if (speed > 10) {
          speed = speed - 2;      // break really hard
     } else if (speed > 0) {
          speed--;                // soft brakes
     } else {
          System.err.println(
              "The bicycle has already stopped!");
     }
}
```

- switch: Equivalent to sequence of chained if else statements



2 Language Basics – Control Flow Statements

- while: do something as long as some condition (boolean expression) is true

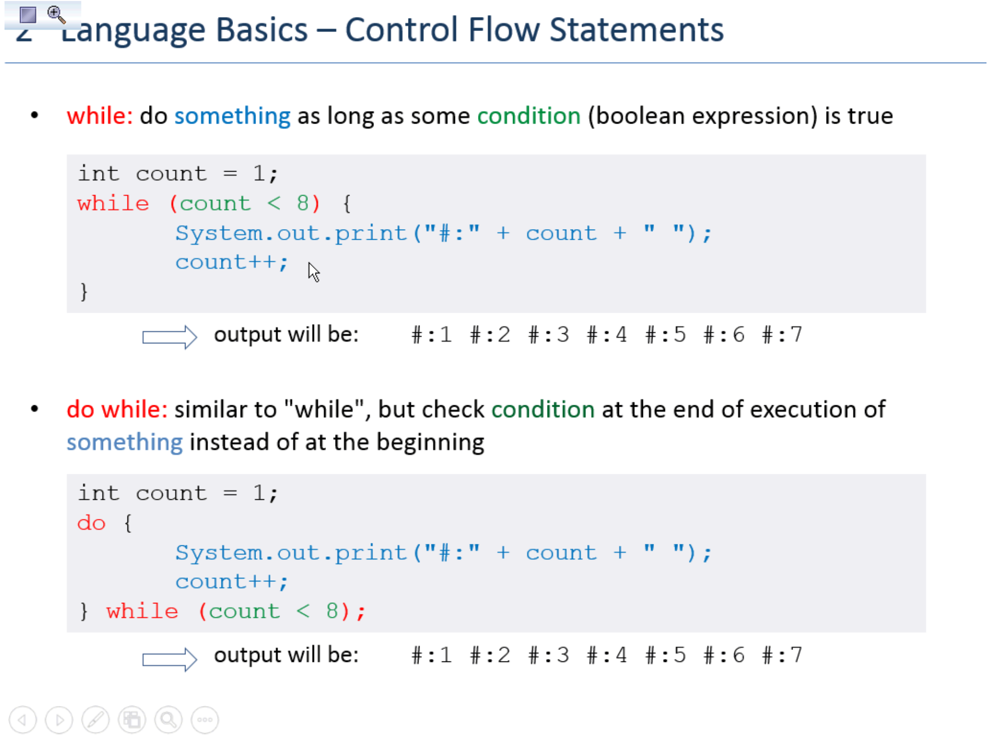
```
int count = 1;
while (count < 8) {
        System.out.print("#:" + count + " ");
        count++;
}
```

⇨ output will be: `#:1 #:2 #:3 #:4 #:5 #:6 #:7`

- do while: similar to "while", but check condition at the end of execution of something instead of at the beginning

```
int count = 1;
do {
        System.out.print("#:" + count + " ");
        count++;
} while (count < 8);
```

⇨ output will be: `#:1 #:2 #:3 #:4 #:5 #:6 #:7`



2 Language Basics – Control Flow Statements

- while: do something as long as some condition (boolean expression) is true

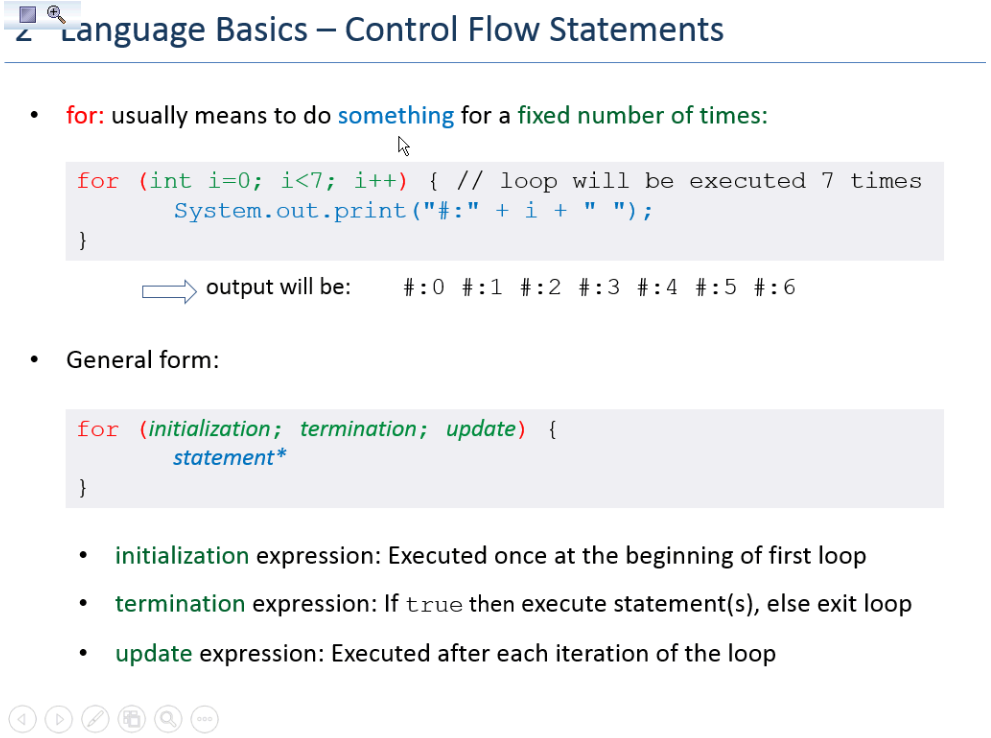
```
int count = 1;
while (count < 8) {
        System.out.print("#:" + count + " ");
        count++;
}
```

⇨ output will be: `#:1 #:2 #:3 #:4 #:5 #:6 #:7`

- do while: similar to "while", but check condition at the end of execution of something instead of at the beginning

```
int count = 1;
do {
        System.out.print("#:" + count + " ");
        count++;
} while (count < 8);
```

⇨ output will be: `#:1 #:2 #:3 #:4 #:5 #:6 #:7`



2 Language Basics – Control Flow Statements

- for: usually means to do something for a fixed number of times:

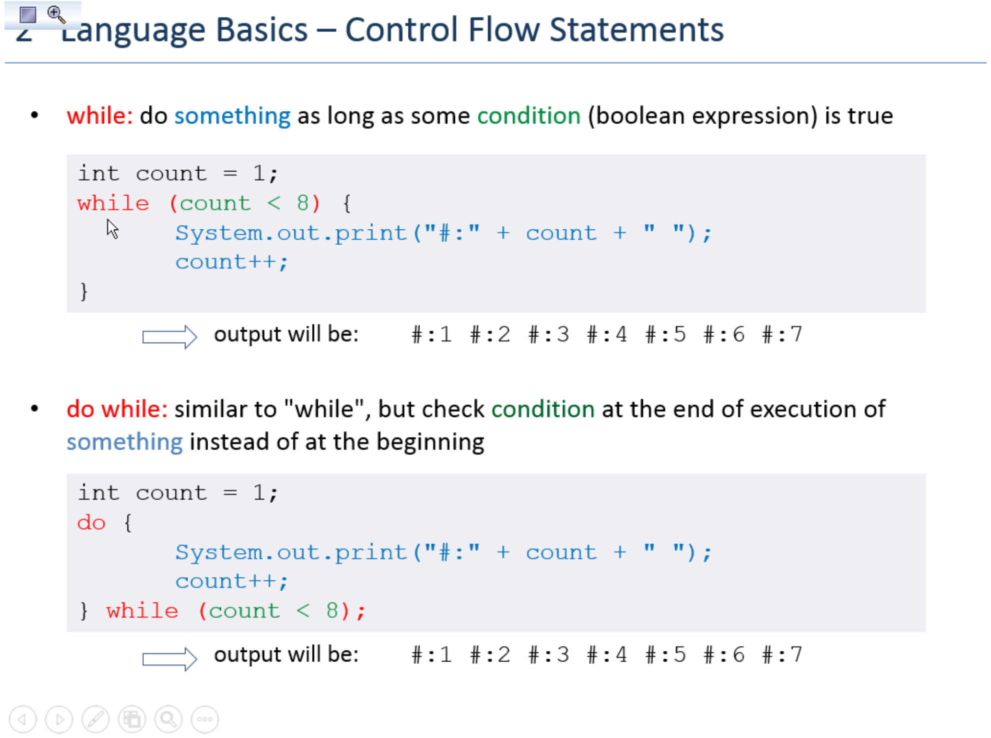
```
for (int i=0; i<7; i++) { // loop will be executed 7 times
        System.out.print("#:" + i + " ");
}
```

⇨ output will be: `#:0 #:1 #:2 #:3 #:4 #:5 #:6`

- General form:

```
for (initialization; termination; update) {
        statement*
}
```

  - initialization expression: Executed once at the beginning of first loop
  - termination expression: If `true` then execute statement(s), else exit loop
  - update expression: Executed after each iteration of the loop



2 Language Basics – Control Flow Statements

- while: do something as long as some condition (boolean expression) is true

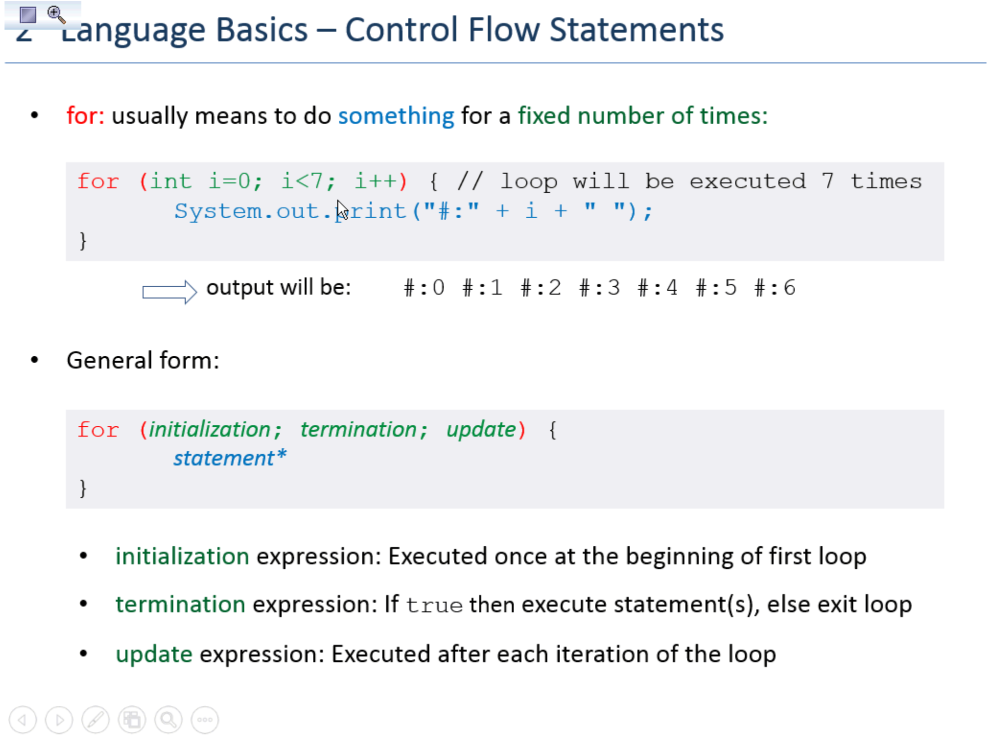
```
int count = 1;
while (count < 8) {
        System.out.print("#:" + count + " ");
        count++;
}
```

⇨ output will be: `#:1 #:2 #:3 #:4 #:5 #:6 #:7`

- do while: similar to "while", but check condition at the end of execution of something instead of at the beginning

```
int count = 1;
do {
        System.out.print("#:" + count + " ");
        count++;
} while (count < 8);
```

⇨ output will be: `#:1 #:2 #:3 #:4 #:5 #:6 #:7`



2 Language Basics – Control Flow Statements

- for: usually means to do something for a fixed number of times:

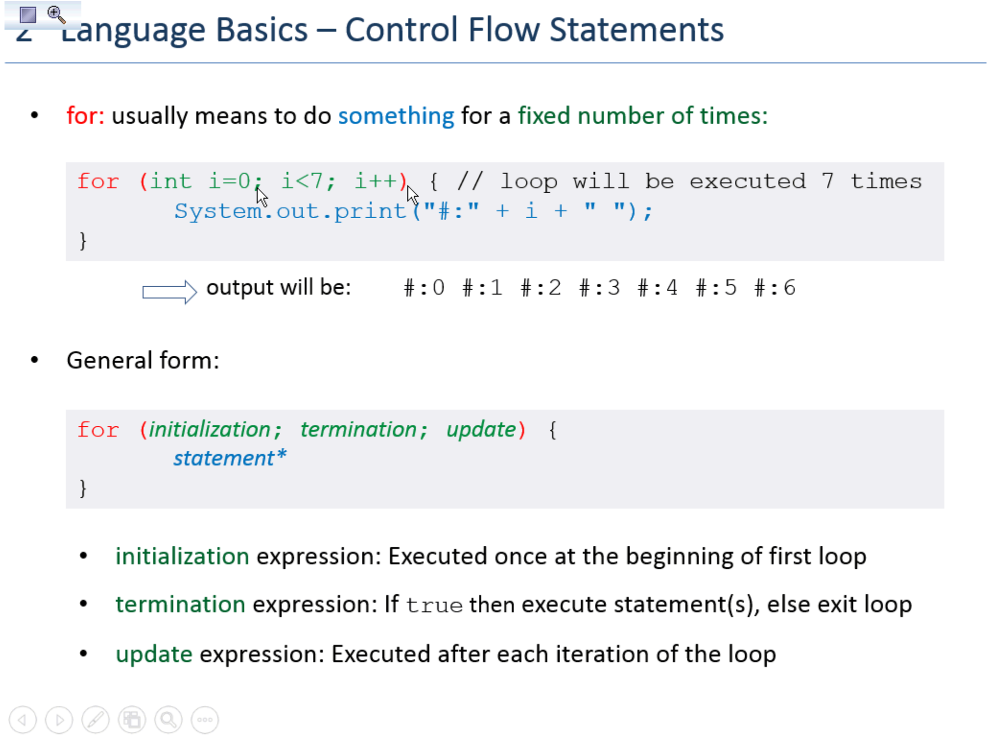
```
for (int i=0; i<7; i++) { // loop will be executed 7 times
        System.out.print("#:" + i + " ");
}
```

⇨ output will be: `#:0 #:1 #:2 #:3 #:4 #:5 #:6`

- General form:

```
for (initialization; termination; update) {
        statement*
}
```

  - initialization expression: Executed once at the beginning of first loop
  - termination expression: If `true` then execute statement(s), else exit loop
  - update expression: Executed after each iteration of the loop



2 Language Basics – Control Flow Statements

- for: usually means to do something for a fixed number of times:

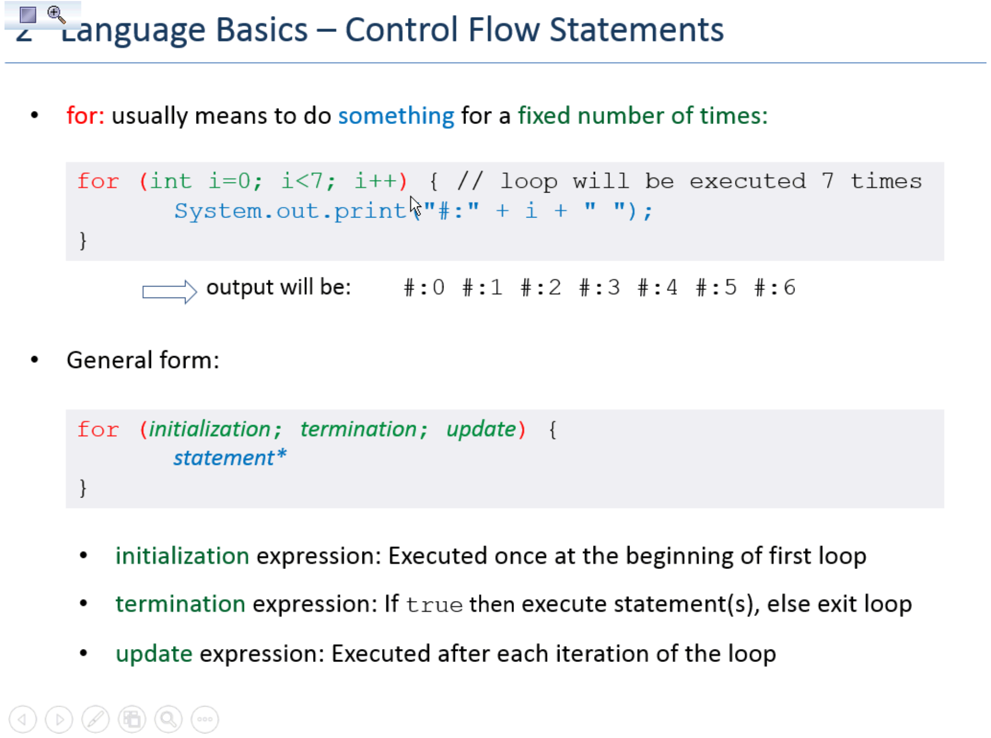
```
for (int i=0; i<7; i++) { // loop will be executed 7 times
        System.out.print("#:" + i + " ");
}
```

⇨ output will be: `#:0 #:1 #:2 #:3 #:4 #:5 #:6`

- General form:

```
for (initialization; termination; update) {
        statement*
}
```

  - initialization expression: Executed once at the beginning of first loop
  - termination expression: If `true` then execute statement(s), else exit loop
  - update expression: Executed after each iteration of the loop



2 Language Basics – Control Flow Statements

- for: usually means to do something for a fixed number of times:

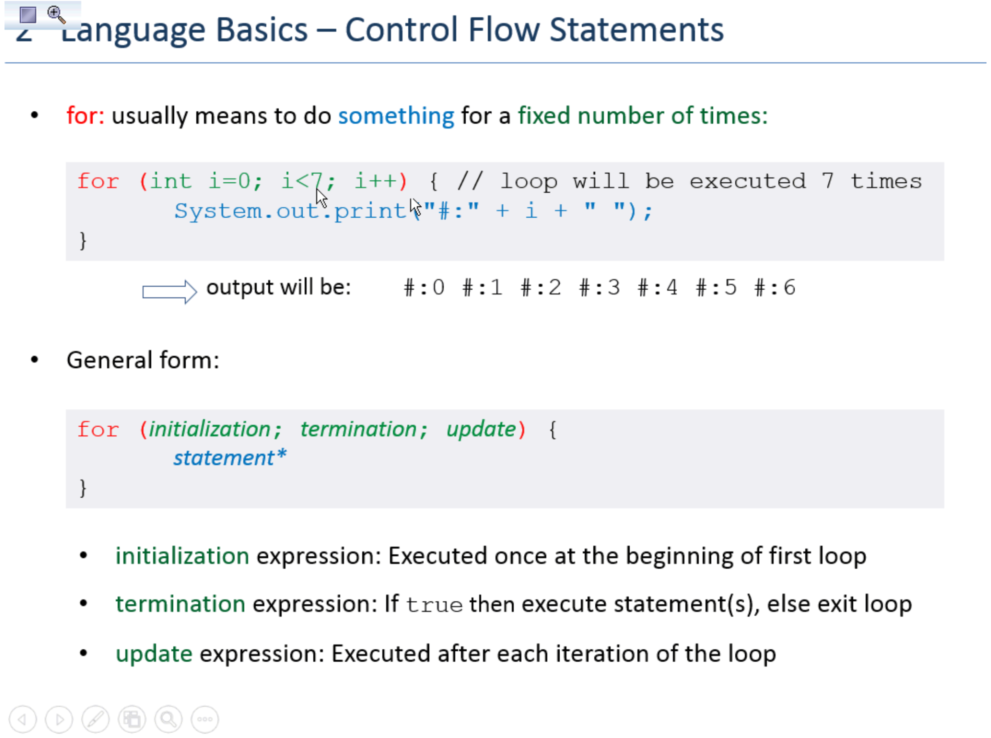
```
for (int i=0; i<7; i++) { // loop will be executed 7 times
        System.out.print("#:" + i + " ");
}
```

⇨ output will be: `#:0 #:1 #:2 #:3 #:4 #:5 #:6`

- General form:

```
for (initialization; termination; update) {
        statement*
}
```

  - initialization expression: Executed once at the beginning of first loop
  - termination expression: If `true` then execute statement(s), else exit loop
  - update expression: Executed after each iteration of the loop



2 Language Basics – Control Flow Statements

- for: usually means to do something for a fixed number of times:

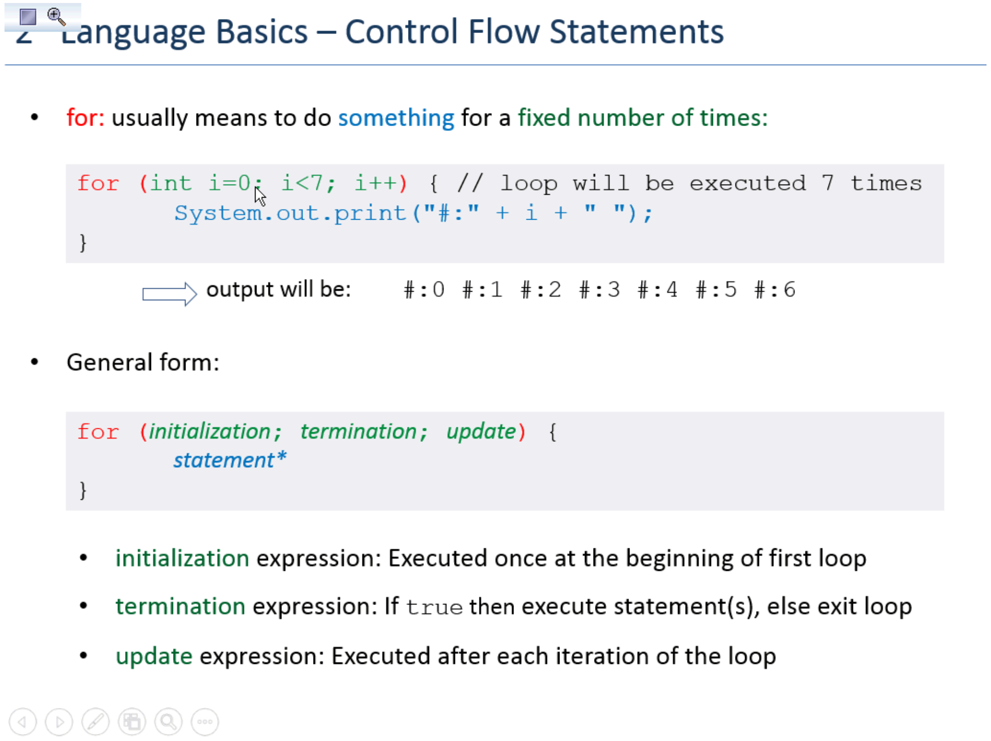
```
for (int i=0; i<7; i++) { // loop will be executed 7 times
        System.out.print("#:" + i + " ");
}
```

⇨ output will be: `#:0 #:1 #:2 #:3 #:4 #:5 #:6`

- General form:

```
for (initialization; termination; update) {
        statement*
}
```

  - initialization expression: Executed once at the beginning of first loop
  - termination expression: If `true` then execute statement(s), else exit loop
  - update expression: Executed after each iteration of the loop



2 Language Basics – Control Flow Statements

- for: usually means to do something for a fixed number of times:

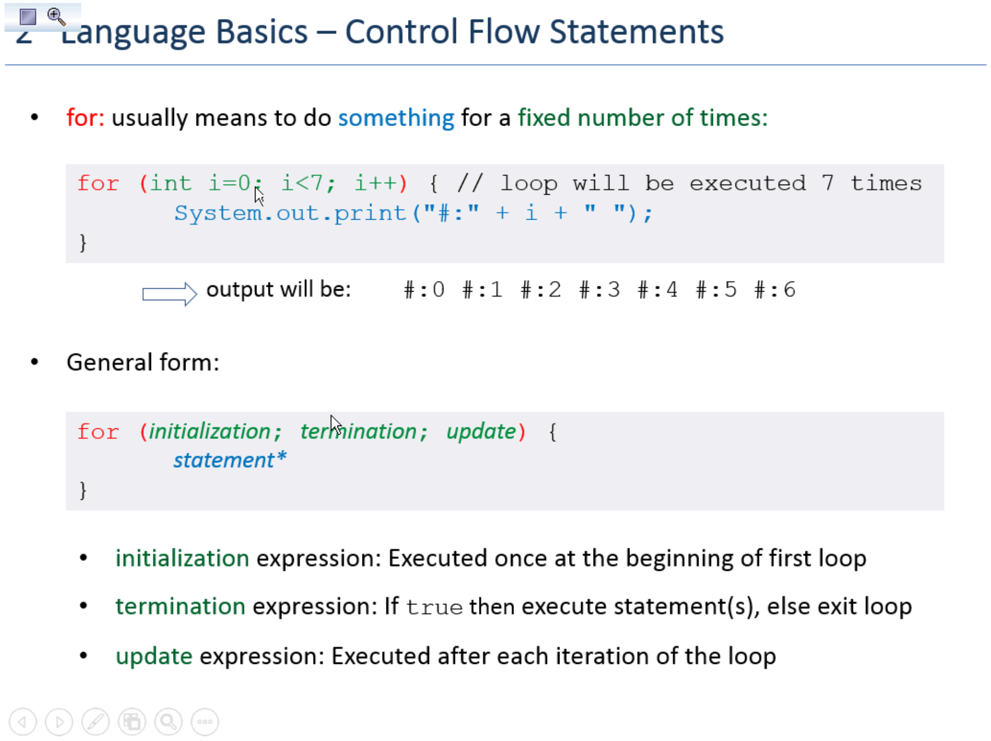
```
for (int i=0; i<7; i++) { // loop will be executed 7 times
       System.out.print("#:" + i + " ");
}
```

⇨ output will be: `#:0 #:1 #:2 #:3 #:4 #:5 #:6`

- General form:

```
for (initialization; termination; update) {
       statement*
}
```

  - initialization expression: Executed once at the beginning of first loop
  - termination expression: If `true` then execute statement(s), else exit loop
  - update expression: Executed after each iteration of the loop



2 Language Basics – Control Flow Statements

- for: usually means to do something for a fixed number of times:

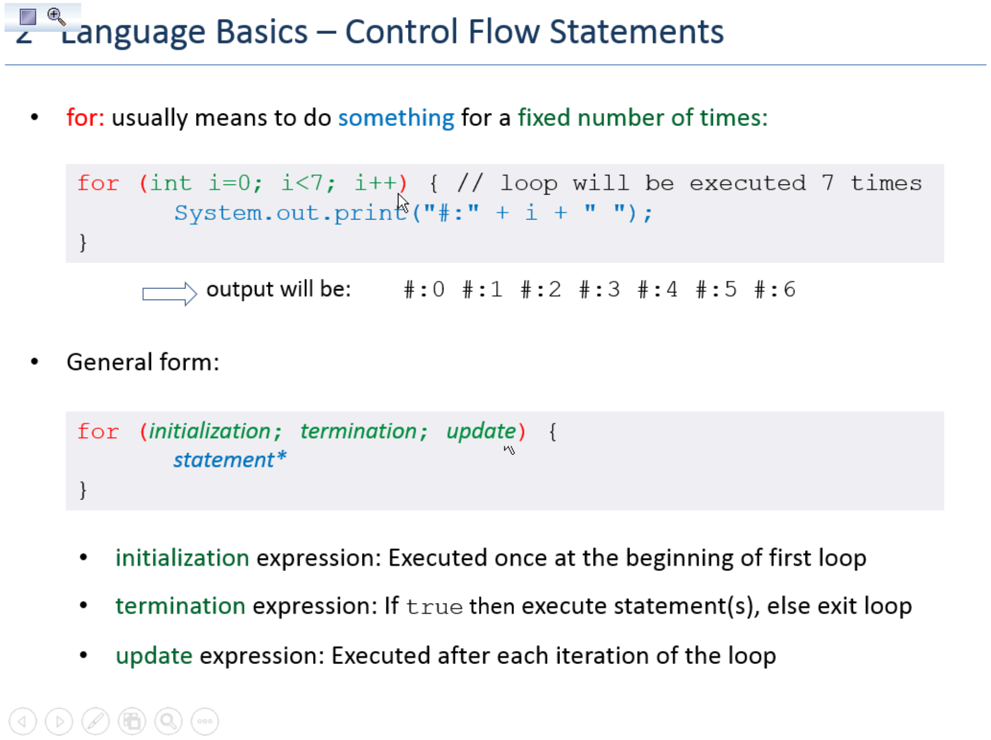
```
for (int i=0; i<7; i++) { // loop will be executed 7 times
       System.out.print("#:" + i + " ");
}
```

⇨ output will be: `#:0 #:1 #:2 #:3 #:4 #:5 #:6`

- General form:

```
for (initialization; termination; update) {
       statement*
}
```

  - initialization expression: Executed once at the beginning of first loop
  - termination expression: If `true` then execute statement(s), else exit loop
  - update expression: Executed after each iteration of the loop



2 Language Basics – Control Flow Statements

- for: usually means to do something for a fixed number of times:

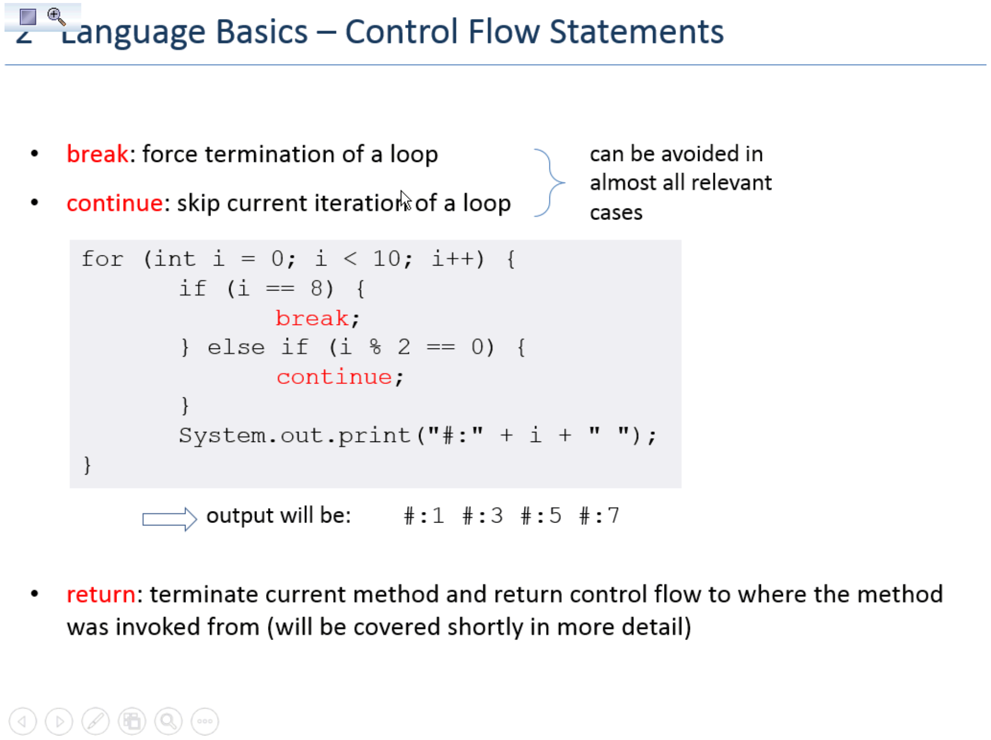
```
for (int i=0; i<7; i++) { // loop will be executed 7 times
       System.out.print("#:" + i + " ");
}
```

⇨ output will be: `#:0 #:1 #:2 #:3 #:4 #:5 #:6`

- General form:

```
for (initialization; termination; update) {
       statement*
}
```

  - initialization expression: Executed once at the beginning of first loop
  - termination expression: If `true` then execute statement(s), else exit loop
  - update expression: Executed after each iteration of the loop



2 Language Basics – Control Flow Statements

- break: force termination of a loop
- continue: skip current iteration of a loop

} can be avoided in almost all relevant cases

```
for (int i = 0; i < 10; i++) {
       if (i == 8) {
              break;
       } else if (i % 2 == 0) {
              continue;
       }
       System.out.print("#:" + i + " ");
}
```

⇨ output will be: `#:1 #:3 #:5 #:7`

- return: terminate current method and return control flow to where the method was invoked from (will be covered shortly in more detail)

2 Language Basics – Control Flow Statements

- break: force termination of a loop
- continue: skip current iteration of a loop

} can be avoided in almost all relevant cases

```
for (int i = 0; i < 10; i++) {
        if (i == 8) {
                break;
        } else if (i % 2 == 0) {
                continue;
        }
        System.out.print("#:" + i + " ");
}
```

output will be: `#:1 #:3 #:5 #:7`

- return: terminate current method and return control flow to where the method was invoked from (will be covered shortly in more detail)

2 Language Basics – Control Flow Statements

- break: force termination of a loop
- continue: skip current iteration of a loop

} can be avoided in almost all relevant cases

```
for (int i = 0; i < 10; i++) {
        if (i == 8) {
                break;
        } else if (i % 2 == 0) {
                continue;
        }
        System.out.print("#:" + i + " ");
}
```

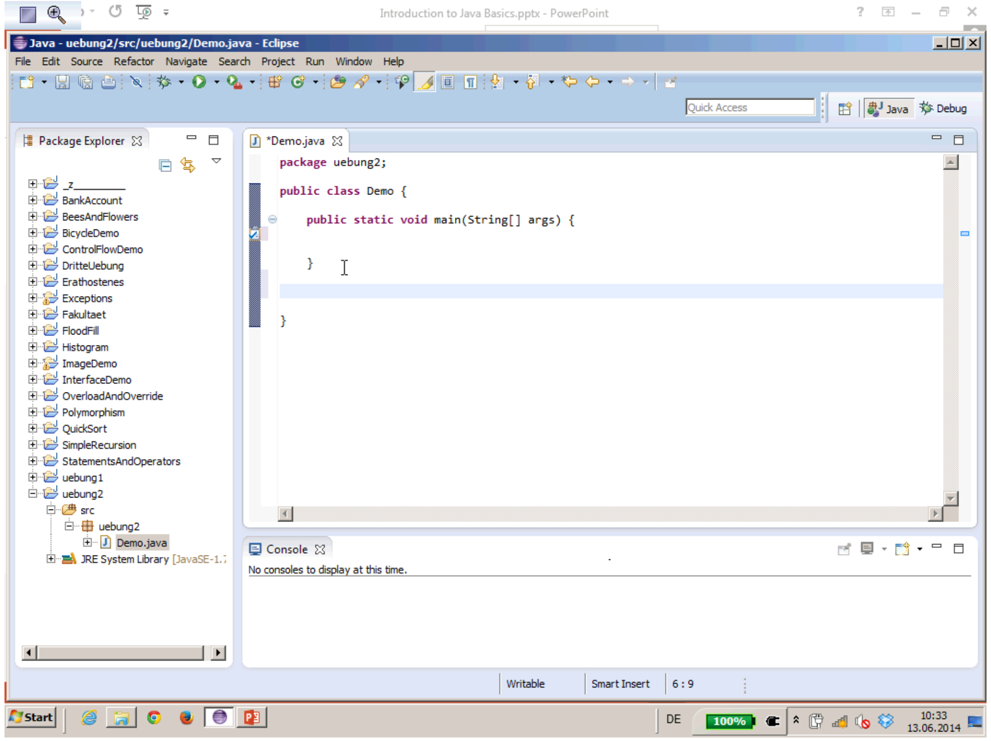
output will be: `#:1 #:3 #:5 #:7`

- return: terminate current method and return control flow to where the method was invoked from (will be covered shortly in more detail)

2 Language Basics – Control Flow Statements

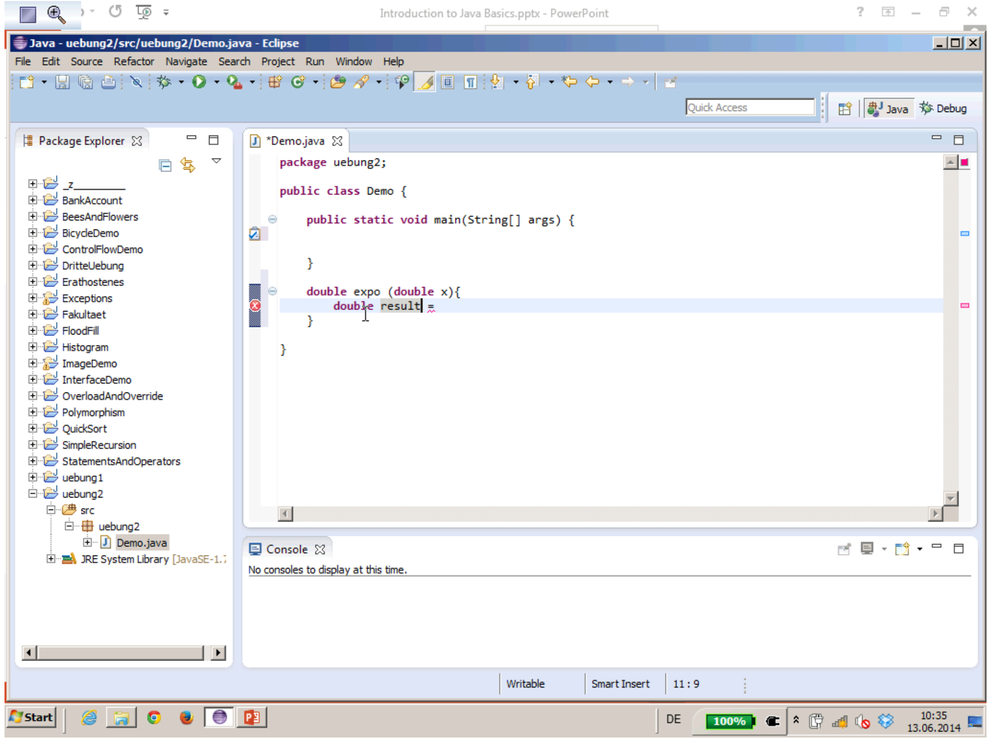
Control Flow Statements

- Control flow statements:
  Allow for deviation of control flow from sequential order of statements:
  - conditionals: `if, if else, switch`
  - loops: `while, do while, for`
  - branches: `break, continue, return`

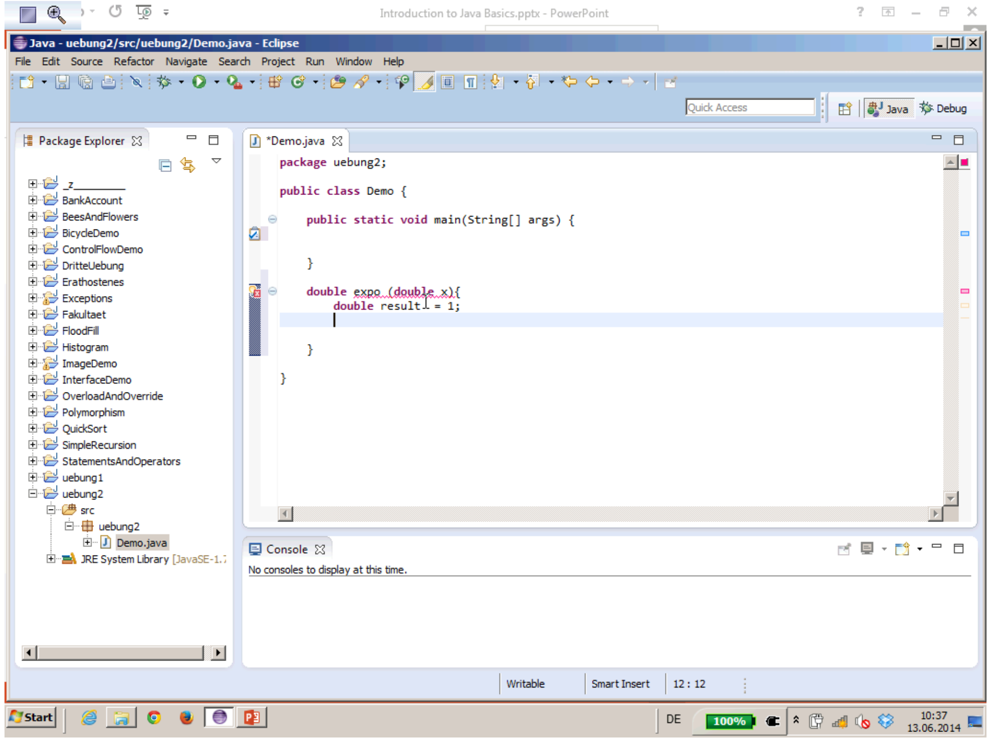


```
package uebung2;

public class Demo {

    public static void main(String[] args) {

    }

}
```

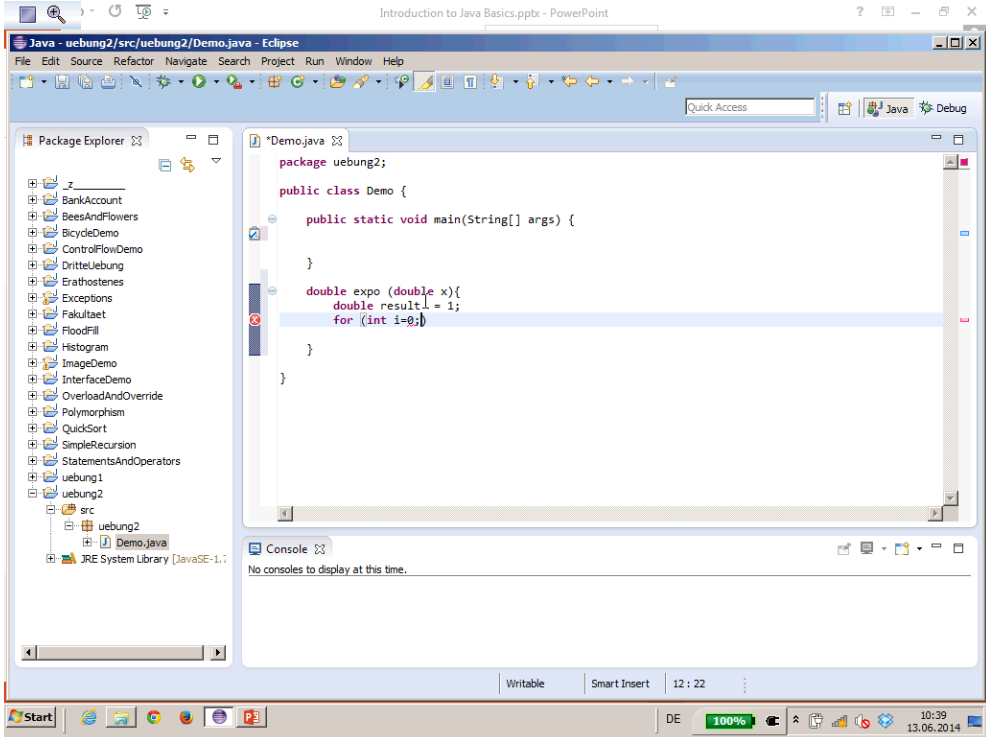
No consoles to display at this time.


Introduction to Java Basics.pptx - PowerPoint
Java - uebung2/src/uebung2/Demo.java - Eclipse
Package Explorer
*Demo.java
Console
No consoles to display at this time.
Writable
Smart Insert
11 : 9
10:35
13.06.2014

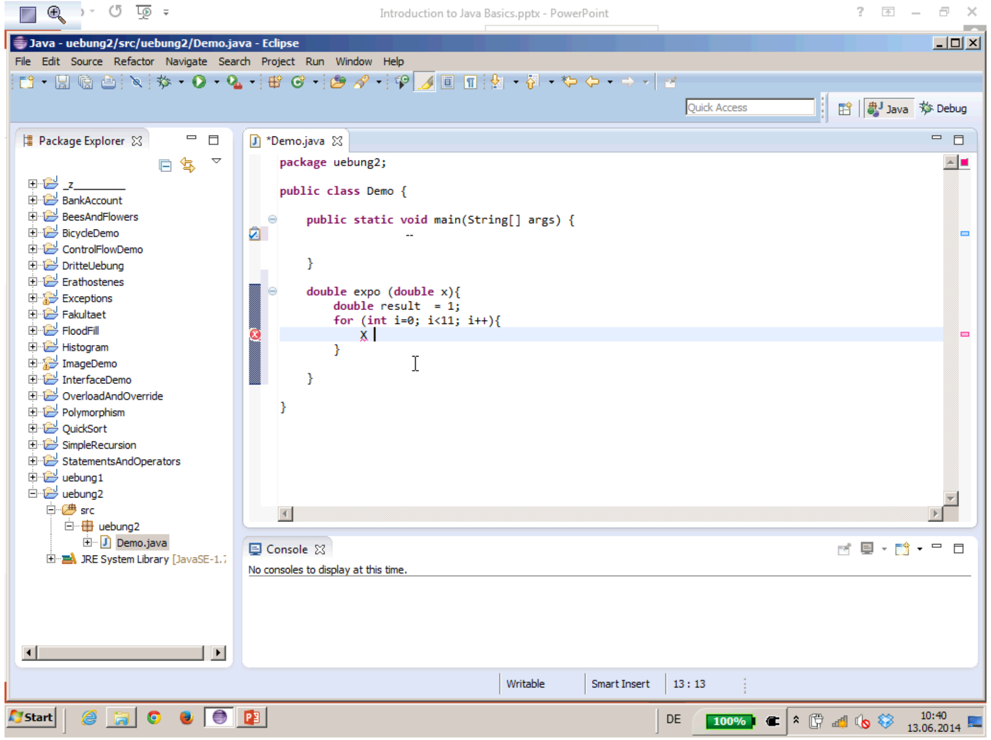
```
package uebung2;

public class Demo {

    public static void main(String[] args) {


    }

    double expo (double x){
        double result =
    }

}
```


Introduction to Java Basics.pptx - PowerPoint
Java - uebung2/src/uebung2/Demo.java - Eclipse
Package Explorer
*Demo.java
Console
No consoles to display at this time.
Writable
Smart Insert
12 : 12
10:37
13.06.2014

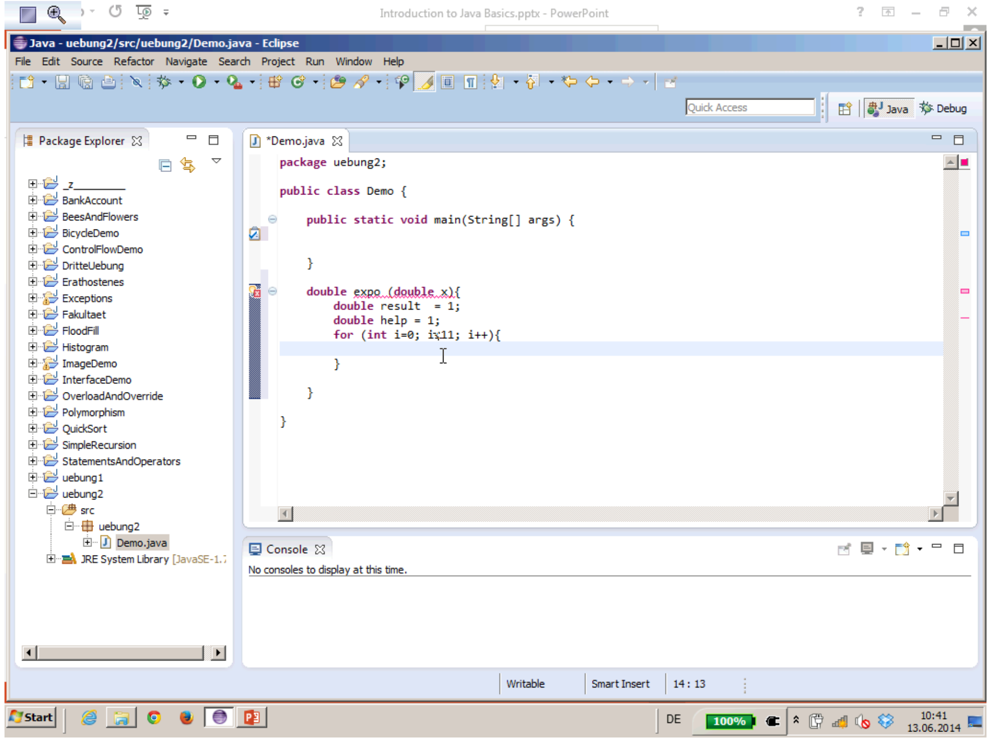
```
package uebung2;

public class Demo {

    public static void main(String[] args) {


    }

    double expo (double x){
        double result = 1;


    }

}
```


Introduction to Java Basics.pptx - PowerPoint
Java - uebung2/src/uebung2/Demo.java - Eclipse
Package Explorer
*Demo.java
Console
No consoles to display at this time.
Writable
Smart Insert
12 : 22
10:39
13.06.2014

```
package uebung2;

public class Demo {

    public static void main(String[] args) {


    }

    double expo (double x){
        double result = 1;
        for (int i=0;)

    }

}
```

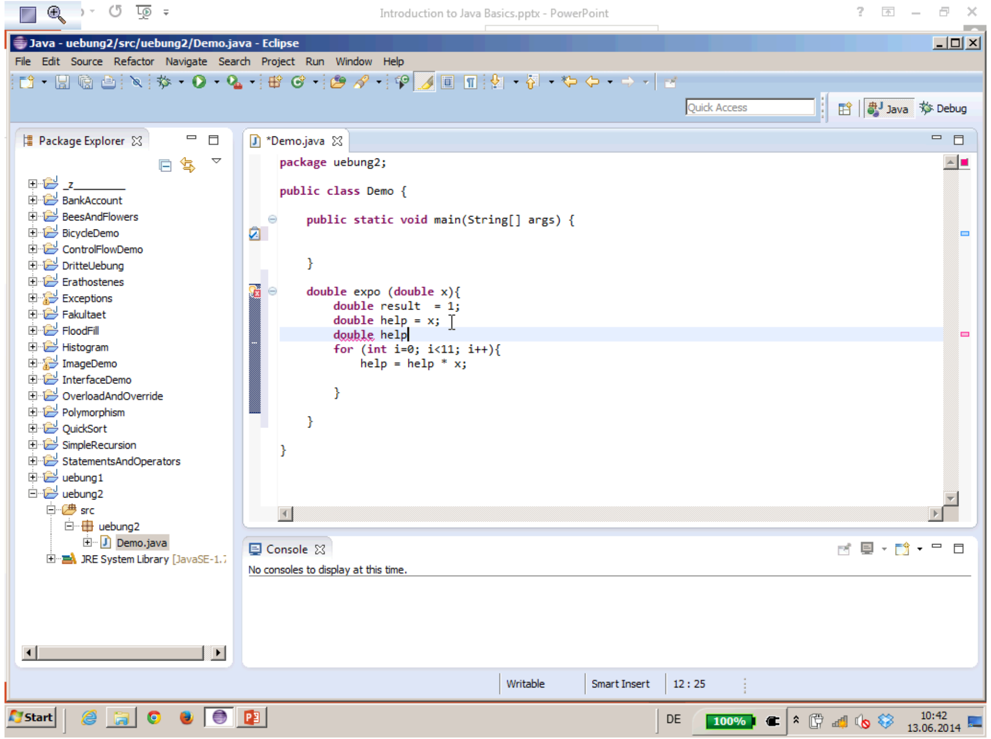

Introduction to Java Basics.pptx - PowerPoint
Java - uebung2/src/uebung2/Demo.java - Eclipse
Package Explorer
*Demo.java
Console
No consoles to display at this time.
Writable
Smart Insert
13 : 13
10:40
13.06.2014

```
package uebung2;

public class Demo {

    public static void main(String[] args) {
                --

    }

    double expo (double x){
        double result  = 1;
        for (int i=0; i<11; i++){
            x
        }

    }

}
```


Introduction to Java Basics.pptx - PowerPoint
Java - uebung2/src/uebung2/Demo.java - Eclipse
File Edit Source Refactor Navigate Search Project Run Window Help
Package Explorer
*Demo.java
Console
Writable
Smart Insert
14 : 13

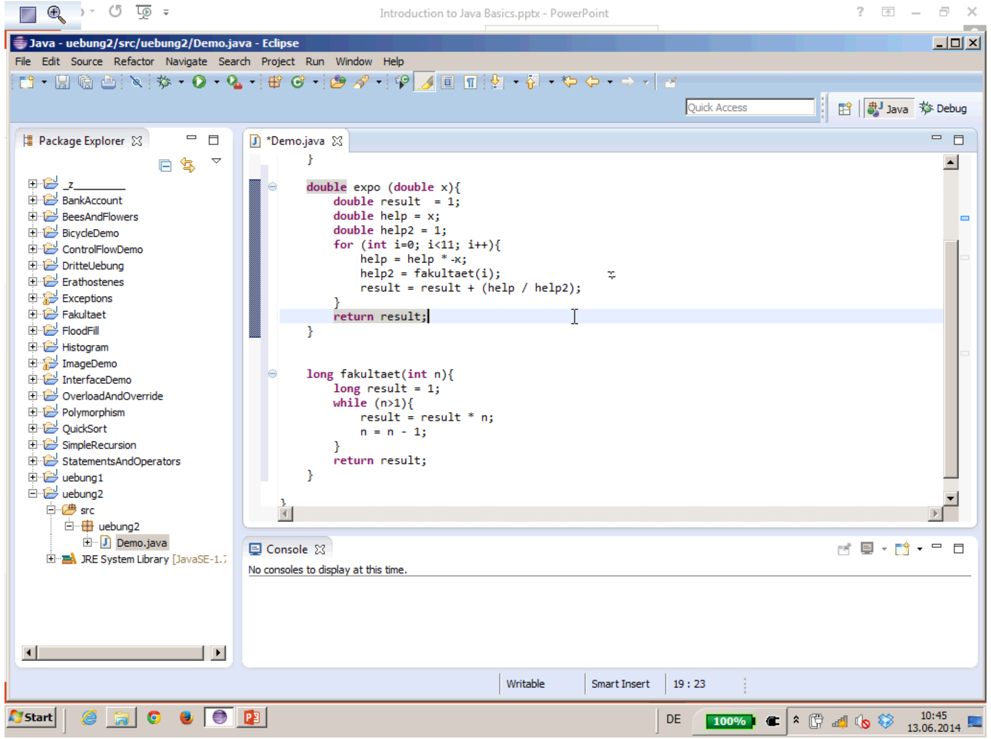
```java
package uebung2;

public class Demo {

    public static void main(String[] args) {


    }

    double expo (double x){
        double result  = 1;
        double help = 1;
        for (int i=0; i<11; i++){

        }

    }

}
```

No consoles to display at this time.


Introduction to Java Basics.pptx - PowerPoint
Java - uebung2/src/uebung2/Demo.java - Eclipse
*Demo.java
Writable
Smart Insert
12 : 25

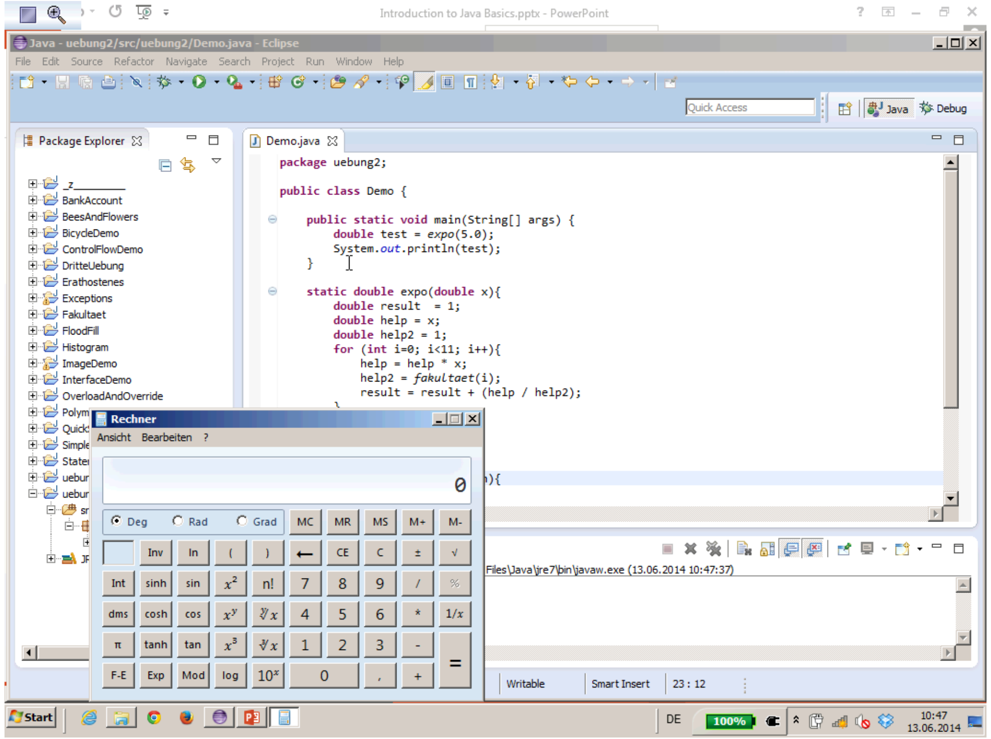
```java
package uebung2;

public class Demo {

    public static void main(String[] args) {


    }

    double expo (double x){
        double result  = 1;
        double help = x;
        double help
        for (int i=0; i<11; i++){
            help = help * x;

        }

    }

}
```

No consoles to display at this time.


Introduction to Java Basics.pptx - PowerPoint
Java - uebung2/src/uebung2/Demo.java - Eclipse
*Demo.java
Writable
Smart Insert
19 : 23

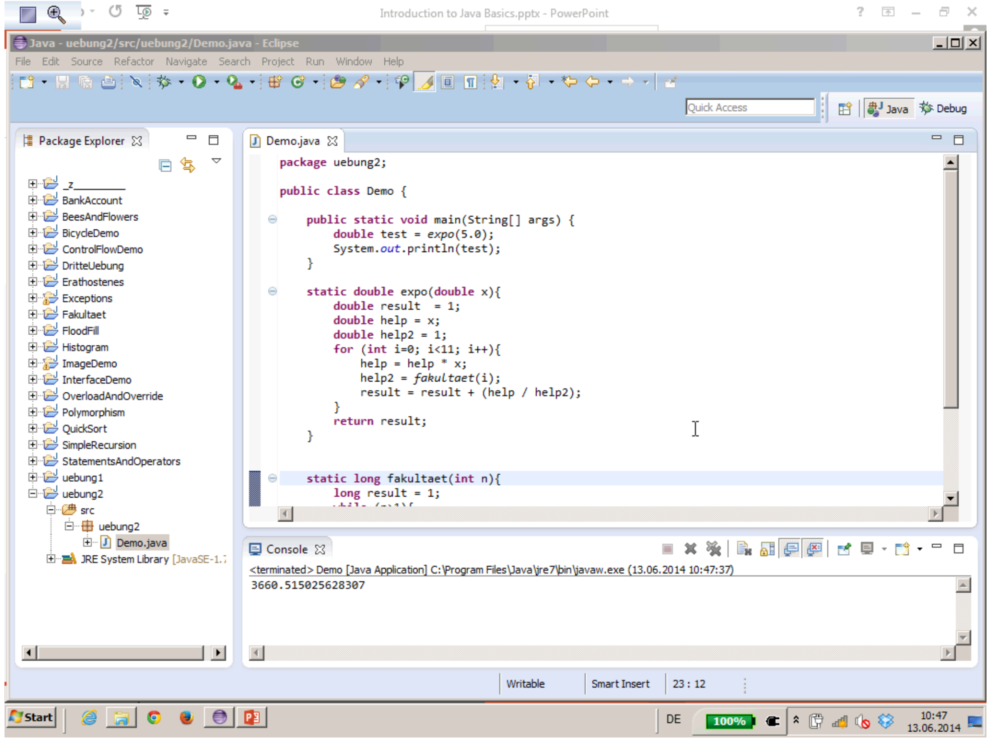
```java
    }

    double expo (double x){
        double result  = 1;
        double help = x;
        double help2 = 1;
        for (int i=0; i<11; i++){
            help = help * -x;
            help2 = fakultaet(i);
            result = result + (help / help2);
        }
        return result;
    }


    long fakultaet(int n){
        long result = 1;
        while (n>1){
            result = result * n;
            n = n - 1;
        }
        return result;
    }

}
```

No consoles to display at this time.


Introduction to Java Basics.pptx - PowerPoint
Java - uebung2/src/uebung2/Demo.java - Eclipse
Demo.java
Rechner
Ansicht Bearbeiten ?
0
Deg Rad Grad
MC MR MS M+ M-
Inv ln ( ) ← CE C ± √
Int sinh sin x² n! 7 8 9 / %
dms cosh cos xʸ 4 5 6 * 1/x
π tanh tan x³ 1 2 3 -
F-E Exp Mod log 10ˣ 0 , + =
Writable
Smart Insert
23 : 12

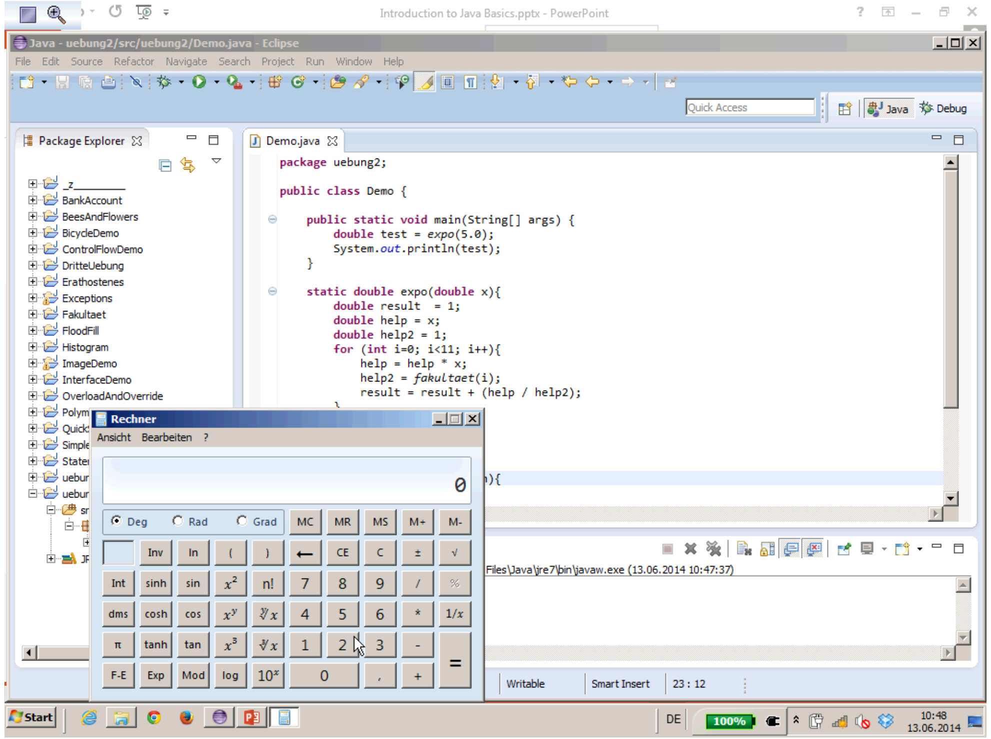
```java
package uebung2;

public class Demo {

    public static void main(String[] args) {
        double test = expo(5.0);
        System.out.println(test);
    }

    static double expo(double x){
        double result  = 1;
        double help = x;
        double help2 = 1;
        for (int i=0; i<11; i++){
            help = help * x;
            help2 = fakultaet(i);
            result = result + (help / help2);
```

Files\Java\jre7\bin\javaw.exe (13.06.2014 10:47:37)



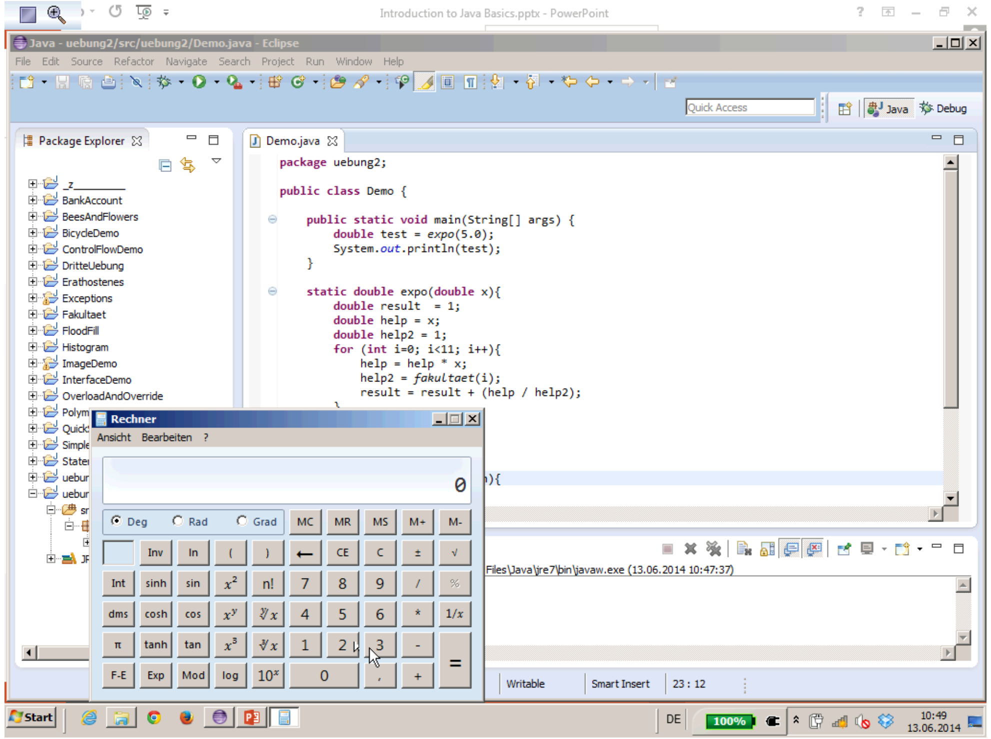
```java
package uebung2;

public class Demo {

    public static void main(String[] args) {
        double test = expo(5.0);
        System.out.println(test);
    }

    static double expo(double x){
        double result  = 1;
        double help = x;
        double help2 = 1;
        for (int i=0; i<11; i++){
            help = help * x;
            help2 = fakultaet(i);
            result = result + (help / help2);
        }
        return result;
    }


    static long fakultaet(int n){
        long result = 1;
```

```
<terminated> Demo [Java Application] C:\Program Files\Java\jre7\bin\javaw.exe (13.06.2014 10:47:37)
3660.515025628307
```


Introduction to Java Basics.pptx - PowerPoint
Java - uebung2/src/uebung2/Demo.java - Eclipse
Rechner
Ansicht Bearbeiten ?
0
○ Deg ○ Rad ○ Grad
MC MR MS M+ M-
Inv ln ( ) ← CE C ± √
Int sinh sin x² n! 7 8 9 / %
dms cosh cos xʸ ʸ√x 4 5 6 * 1/x
π tanh tan x³ ∛x 1 2 3 -
F-E Exp Mod log 10ˣ 0 , + =
Writable
Smart Insert
23 : 12
10:48
13.06.2014

```java
package uebung2;

public class Demo {

    public static void main(String[] args) {
        double test = expo(5.0);
        System.out.println(test);
    }

    static double expo(double x){
        double result  = 1;
        double help = x;
        double help2 = 1;
        for (int i=0; i<11; i++){
            help = help * x;
            help2 = fakultaet(i);
            result = result + (help / help2);
```

```
Files\Java\jre7\bin\javaw.exe (13.06.2014 10:47:37)
```


Introduction to Java Basics.pptx - PowerPoint
Java - uebung2/src/uebung2/Demo.java - Eclipse
Rechner
Ansicht Bearbeiten ?
0
○ Deg ○ Rad ○ Grad
MC MR MS M+ M-
Inv ln ( ) ← CE C ± √
Int sinh sin x² n! 7 8 9 / %
dms cosh cos xʸ ʸ√x 4 5 6 * 1/x
π tanh tan x³ ∛x 1 2 3 -
F-E Exp Mod log 10ˣ 0 , + =
Writable
Smart Insert
23 : 12
10:49
13.06.2014

```java
package uebung2;

public class Demo {

    public static void main(String[] args) {
        double test = expo(5.0);
        System.out.println(test);
    }

    static double expo(double x){
        double result  = 1;
        double help = x;
        double help2 = 1;
        for (int i=0; i<11; i++){
            help = help * x;
            help2 = fakultaet(i);
            result = result + (help / help2);
```

```
Files\Java\jre7\bin\javaw.exe (13.06.2014 10:47:37)
```