# Java -Classes

## Class

A basis for the Java language.

Each concept we wish to describe in Java must be included inside a class.

A class defines a new data type, whose values are objects:

1) a class is a template for objects
2) an object is an instance of a class

## Class Definition

A class contains a **name**, several **variable** declarations (instance variables) and several **method** declarations. All are called **members** of the class.

General form of a class:

```
class classname {

type instance-variable-1;
 ...
 type instance-variable-n;

type method-name-1(parameter-list) { ... }
type method-name-2(parameter-list) { ... }
...
type method-name-m(parameter-list) { ... }
}
```

## Example: Class

A class with three variable members:

```
 class Box {
double width;
 double height;
 double depth;
 }
```

A new Box object is created and a new value assigned to its width variable:

```
 Box myBox = new Box();
myBox.width = 100;
```

Talent Accelerator
powered by START

## Example: Class Usage

```
class BoxDemo {
 public static void main(String args[]) {
Box mybox = new Box();
double vol;
mybox.width = 10;
mybox.height = 20;
 mybox.depth = 15;

vol = mybox.width * mybox.height * mybox.depth;
 System.out.println("Volume is " + vol);
          }
 }
```

zalando

## Compilation and Execution

Place the *Box* class definitions in file *Box.java:*

```
class Box { … }
```

Place the *BoxDemo* class definitions in file *BoxDemo.java:*

```
class BoxDemo {
 public static void main(…) { … }
}
```

Compilation and execution:

```
> javac BoxDemo.java

> java BoxDemo
```

## Variable Independence 1

Each object has its own copy of the instance variables: changing the variables of one object has no effect on the variables of another object.

Consider this example:

```
class BoxDemo2 {
public static void main(String args[]) {

Box mybox1 = new Box();

Box mybox2 = new Box();

double vol;

mybox1.width = 10;

mybox1.height = 20;

mybox1.depth = 15;
```

## Variable Independence 2

```
mybox2.width = 3;
mybox2.height = 6;
mybox2.depth = 9;

vol = mybox1.width * mybox1.height * mybox1.depth;
System.out.println("Volume is " + vol);

vol = mybox2.width * mybox2.height * mybox2.depth;
 System.out.println("Volume is " + vol);
 }
}
```

What are the printed volumes of both boxes?

## Declaring Objects

Obtaining objects of a class is a two-stage process:

1) Declare a variable of the class type:

```
Box myBox;
```

The value of *myBox* is a reference to an object, if one exists, or null. At this moment, the value of *myBox* is *null*.

2) Acquire an actual, physical copy of an object and assign its address to the variable.

## Operator new

Allocates memory for a Box object and returns its address:

```
Box myBox = new Box();
```

The address is then stored in the *myBox* reference variable.

*Box()* is a class constructor - a class may declare its own constructor or rely on the default constructor provided by the Java environment.

## Memory Allocation

Memory is allocated for objects dynamically.

This has both advantages and disadvantages:

1) as many objects are created as needed

2) allocation is uncertain – memory may be insufficient

Variables of simple types do not require new:

```
int n = 1;
```

In the interest of efficiency, Java does not implement simple types as objects. Variables of simple types hold values, not references.

## Assigning Reference Variables

Assignment copies address, not the actual value:

```
Box b1 = new Box();
Box b2 = b1;
```

Both variables point to the same object.

Variables are not in any way connected. After

```
b1 = null;
```

b2 still refers to the original object.

## Methods

General form of a method definition:

```
type name(parameter-list) {
... return value; ...
}
```

Components:

1) type - type of values returned by the method. If a method does not return any value, its return type must be void.
2) name is the name of the method
3) parameter-list is a sequence of type-identifier lists separated by commas
4) return value indicates what value is returned by the method.

## Constructor

A constructor initializes the instance variables of an object.

It is called immediately after the object is created but before the new operator completes.

1) it is syntactically similar to a method:

2) it has the same name as the name of its class

3) it is written without return type; the default return type of a class constructor is the same class

When the class has no constructor, the default constructor automatically initializes all its instance variables with zero.

Talent Accelerator
powered by START

## finalize() Method

A constructor helps to initialize an object just after it has been created.

In contrast, the finalize method is invoked just before the object is destroyed:

1) implemented inside a class as:

```
protected void finalize() { … }
```

2) implemented when the usual way of removing objects from memory is insufficient, and some special actions has to be carried out

zalando

Talent Accelerator
powered by START

## Garbage Collection

Garbage collection is a mechanism to remove objects from memory when they are no longer needed.

Garbage collection is carried out by the garbage collector:

1) The garbage collector keeps track of how many references an object has.
2) It removes an object from memory when it has no longer any references.
3) Thereafter, the memory occupied by the object can be allocated again.
4) The garbage collector invokes the finalize method

zalando

## Break 10 minutes

## Keyword this

Keyword this allows a method to refer to the object that invoked it.

It can be used inside any method to refer to the current object:

```
Box(double width, double height, double depth) {
this.width = width;
this.height = height;
this.depth = depth;
}
```

The above use of this is redundant but correct

Talent Accelerator
powered by START

## Instance Variable Hiding

Variables with the same names:

1) it is illegal to declare two local variables with the same name inside the same or enclosing scopes
2) it is legal to declare local variables or parameters with the same name as the instance variables of the class.

As the same-named local variables/parameters will hide the instance variables, using this is necessary to regain access to them:

```
Box(double width, double height, double depth) {
this.width = width;
 this.height = height;
this.depth = depth;
}
```

zalando

Talent Accelerator powered by START

## Method overloading

It is legal for a class to have two or more methods with the same name.

However, Java has to be able to uniquely associate the invocation of a method with its definition relying on the number and types of arguments.

Therefore the same-named methods must be distinguished:

1) by the number of arguments, or
2) by the types of arguments

Overloading and inheritance are two ways to implement polymorphism.

zalando

## Construction overloading

Why overload constructors?

Consider this:

```
class Box {
double width, height, depth;

Box(double w, double h, double d) {
width = w; height = h; depth = d;
}
 double volume() {
return width * height * depth;
}
 }
```

All Box objects can be created in one way: passing all three dimensions.

# Argument-Passing

Two types of variables:

1) simple types
2) class types

Two corresponding ways of how the arguments are passed to methods:

1) by value  a method receives a cope of the original value; parameters of simple types
2) by reference  a method receives the memory address of the original value, not the value itself; parameters of class types

## Static Class members

Normally, the members of a class (its variables and methods) may be only used through the objects of this class.

Static members are independent of the objects:

1) variables
2) methods
3) initialization block

All declared with the static keyword.

## Static Variables

Static variable:

```
static int a;
```

Essentially, it a global variable shared by all instances of the class.

It cannot be used within a non-static method

# Static Methods

Static method:

```
static void meth() { … }
```

Several restrictions apply:

- can only call static methods
- must only access static variables
- cannot refer to this or super in any way

## Static Block

Static block:

```
static { … }
```

This is where the static variables are initialized.

The block is executed exactly once, when the class is first loaded.

# Nested Classes

It is possible to define a class within a class – nested class.

The scope of the nested class is its enclosing class: if class B is defined within class A then B is known to A but not outside.

Access rights:

1) a nested class has access to all members of its enclosing class, including its private members

2) the enclosing class does not have access to the members of the nested class

## Types of Nested Classes

There are two types of nested classes:

1) static – cannot access the members of its enclosing class directly, but through an object; defined with the static keyword
2) non-static – has direct access to all members of the enclosing class in the same way as other non-static member of this class so

A static nested class is seldom used.

A non-static nested class is also called an inner class

## Questions?

Talent Accelerator powered by START STEPS

Thank you for your attention!