# 02 - Java Object Orientation

## Join Google +community

## http://goo.gl/U7qVS

**You can ask all your doubts, questions and queries by posting on this G+ community during/after webinar**

# Outline

- Classes and Methods
- Access Modifiers
- Constructors and Overloading
- Inheritence
- Interface
- Non Access Modifiers (Abstract, Final and Static)

# Simple Class and Method

```
Class Fruit{
    int grams;
    int cals_per_gram;

    int total_calories() {
        return(grams*cals_per_gram);
    }
}
```

# Methods

- A method is a named sequence of code that can be invoked by other Java code.
- A method takes some parameters, performs some computations and then optionally returns a value (or object).
- Methods can be used as part of an expression statement.

```
public float convertCelsius(float tempC) {
      return( ((tempC * 9.0f) / 5.0f) + 32.0 );
   }
```

# Method Signatures

- A method signature specifies:
  - The name of the method.
  - The type and name of each parameter.
  - The type of the value (or object) returned by the method.
  - The checked exceptions thrown by the method.
  - Various method modifiers.
  - *modifiers type name ( parameter list ) [throws exceptions ]*

```
public float convertCelsius (float tCelsius ) {}
public boolean setUserInfo ( int i, int j, String name ) throws
    IndexOutOfBoundsException {}
```

# Access Control

- ***public*** member (function/data)
  - Can be called/modified from outside.
- ***protected***
  - Can be called/modified from derived classes
- ***private***
  - Can be called/modified only from the current class
- ***default ( if no access modifier stated )***
  - Usually referred to as “Friendly”.
  - Can be called/modified/instantiated from the same package.

# Public/private

- Methods/data may be declared ***public*** or ***private*** meaning they may or may not be accessed by code in other classes …
- Good practice:
  - keep data private
  - keep most methods private
- well-defined interface between classes - helps to eliminate errors

# Using objects

- Here, code in one class creates an instance of another class and does something with it …

```
Fruit plum=new Fruit();
int cals;
cals = plum.total_calories();
```

- ***Dot operator*** allows you to access (public) data/methods inside Fruit class

# Constructors

- The line

  plum = new Fruit();

- invokes a constructor method with which you can set the initial data of an object
- You may choose several different type of constructor with different argument lists

  eg Fruit(), Fruit(a) ...

# Overloading

- Can have several versions of a method in class with different types/numbers of arguments

  Fruit() {grams=50;}

  Fruit(a,b) { grams=a; cals_per_gram=b;}

- By looking at arguments Java decides which version to use

# String is an Object

- Constant strings as in C, does not exist

- The function call `foo("Hello")` creates a String object, containing “Hello”, and passes reference to it to `foo`.

- There is no point in writing :

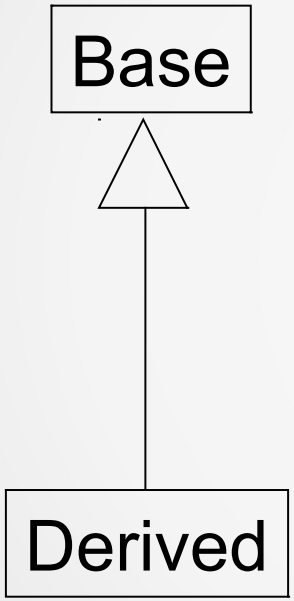
```
String s = new String("Hello");
```

- The String object is a constant. It can’t be changed using a reference to it.

# Packages

- Java code has hierarchical structure.
- The environment variable CLASSPATH contains the directory names of the roots.
- Every Object belongs to a package ( ‘package’ keyword)
- Object full name contains the name full name of the package containing it.

# Inheritance




```
class Base {
    Base(){}
    Base(int i) {}
    protected void foo() {…}
}

class Derived extends Base {
    Derived() {}
    protected void foo() {…}
    Derived(int i) {
      super(i);
       …
      super.foo();
    }
}
```

# Inheritence (2)

- Inheritance creates an “is a” relation:

For example, if B inherits from A, than we say that “B is also an A”.

Implications are:

- access rights (Java forbids reducing access rights) - derived class can receive all the messages that the base class can.
- behavior
- precondition and postcondition

# Inheritance (3)

- In Java, all methods are virtual :

```
class Base {
  void foo() {
    System.out.println("Base");
  }
}
class Derived extends Base {
  void foo() {
    System.out.println("Derived");
  }
}
public class Test {
  public static void main(String[] args) {
    Base b = new Derived();
    b.foo();  // Derived.foo() will be activated
  }
}
```

# Inheritance (4) - Optional

```
class classC extends classB {
    classC(int arg1, int arg2){
      this(arg1);
      System.out.println("In classC(int arg1, int arg2)");
    }
    classC(int arg1){
      super(arg1);
      System.out.println("In classC(int arg1)");
    }
}
class classB extends classA {
    classB(int arg1){
      super(arg1);
      System.out.println("In classB(int arg1)");
    }
    classB(){
      System.out.println("In classB()");
    }
}
```

# Interface

Interfaces are useful for the following:

- Capturing similarities among unrelated classes without artificially forcing a class relationship.
- Declaring methods that one or more classes are expected to implement.
- Revealing an object's programming interface without revealing its class.

# Interface

```
interface IChef {
    void cook(Food food);
}
```

```
interface BabyKicker {
    void kickTheBaby(Baby);
}
```

```
interface SouthParkCharacter {
    void curse();
}
```

```
class Chef implements IChef, SouthParkCharacter {
        // overridden methods MUST be public
        // can you tell why ?
        public void curse() { … }
        public void cook(Food f) { … }
}
```

* access rights (Java forbids reducing of access rights)

# Abstract

- ***abstract*** member function, means that the function does not have an implementation.
- ***abstract*** class, is class that can not be instantiated.

```
AbstractTest.java:6: class AbstractTest is an abstract class.
It can't be instantiated.
        new AbstractTest();
        ^
1 error
```

NOTE:
An abstract class is not required to have an abstract method in it.
But any class that has an abstract method in it or that does not provide an implementation for any abstract methods declared
in its superclasses must be declared as an abstract class.

# Abstract - Example

```
package java.lang;
public abstract class Shape {
       public abstract void draw();
       public void move(int x, int y) {
          setColor(BackGroundColor);
          draw();
          setCenter(x,y);
          setColor(ForeGroundColor);
          draw();
        }
}
```

```
package java.lang;
public class Circle extends Shape {
       public void draw() {
         // draw the circle ...
       }
}
```

# final

- ***final* member data**
  Constant member
- ***final* member function**
  The method can't be overridden.
- ***final* class**
  'Base' is final, thus it can't be extended

(String class is final)

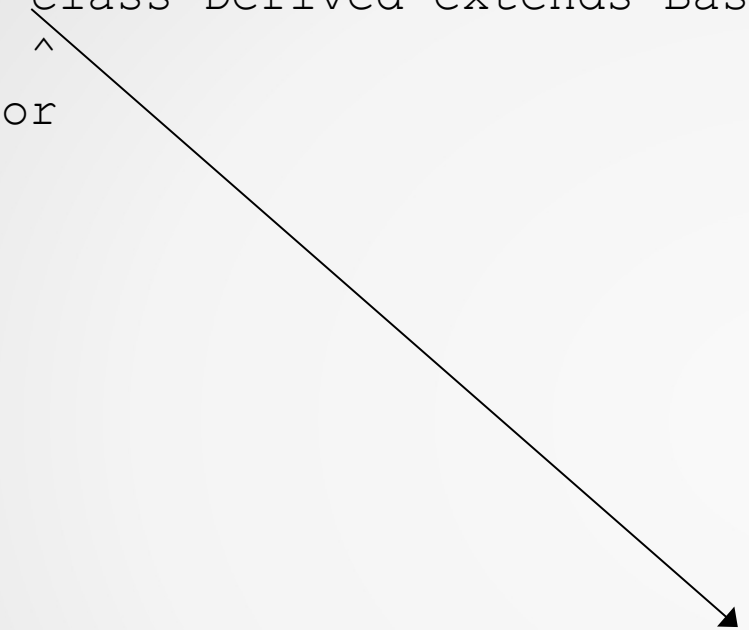
```
final class Base {
  final int i=5;
  final void foo() {
    i=10;
//what will the compiler say
about this?
  }
}

class Derived extends Base
{ // Error
  // another foo ...
  void foo() {

  }
}
```

# final

```
Derived.java:6: Can't subclass final classes: class Base
class class Derived extends Base {
      ^
1 error
```



```
final class Base {
  final int i=5;
  final void foo() {
    i=10;
  }
}

class Derived extends Base
{ // Error
  // another foo ...
  void foo() {

  }
}
```

# Static - [1/4]

- **Member data** - Same data is used for all the instances (objects) of some Class.

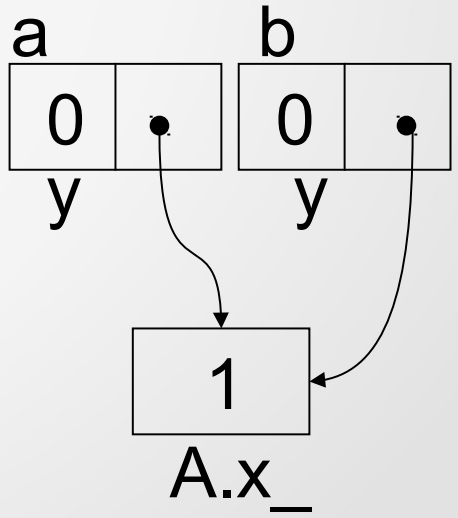
```
Class A {
    public int y = 0;
    public static int x_ = 1;
};

A a = new A();
A b = new A();
System.out.println(b.x_);
a.x_ = 5;
System.out.println(b.x_);
A.x_ = 10;
System.out.println(b.x_);
```

*Assignment performed on the first access to the Class.*
*Only one instance of 'x' exists in memory*

**Output:**

```
1
5
10
```



# Static - [2/4]

- **Member function**
  - Static member function can access only static members
  - Static member function can be called without an instance.

```
Class TeaPot {
        private static int numOfTP = 0;
        private Color myColor_;
        public TeaPot(Color c) {
                myColor_ = c;
                numOfTP++;
        }
        public static int howManyTeaPots()
                { return numOfTP; }

        // error :
        public static Color getColor()
                { return myColor_; }
}
```

# Static - [2/4] cont.

**Usage:**

```
TeaPot tp1 = new TeaPot(Color.RED);

TeaPot tp2 = new TeaPot(Color.GREEN);

System.out.println("We have " +
     TeaPot.howManyTeaPots()+ "Tea Pots");
```

# Static - [3/4]

- **Block**
  - Code that is executed in the first reference to the class.
  - Several static blocks can exist in the same class ( Execution order is by the appearance order in the class definition ).
  - Only static members can be accessed.

```
class RandomGenerator {
   private static int seed_;

   static {
        int t = System.getTime() % 100;
        seed_ = System.getTime();
        while(t-- > 0)
          seed_ = getNextNumber(seed_);
        }
   }
}
```

# Doubts, Questions, Queries?

# Thank You!! :)

- **Java**
  - http://docs.oracle.com/javase/tutorial/

- **Open Android Learning**
  - **Like us on Facebook : - https://www.facebook.com/OpenAndroidLearning**
  - **G+ Community :- http://goo.gl/U7qVS**
  - **Youtube Channel : - http://goo.gl/3pj7D**