Programming Fundamentals

Representing OO Concepts in Java II

Week 7| Iresh Bandara

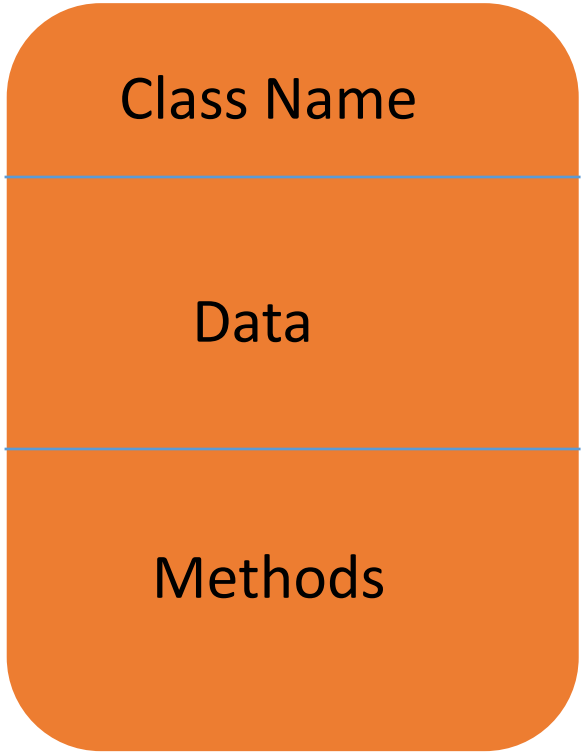
ROBERT GORDON UNIVERSITY ABERDEEN
TEF Gold
INFORMATICS INSTITUTE OF TECHNOLOGY

Learning Outcomes

- Covers part of LO3 & LO4 for Module

- On completion of this lecture, students are expected to be able to:
  - Identify the need for object-oriented concepts in a program.
  - Build simple java applications using object-oriented concepts.
  - Make use of abstract classes in java programs.

Structure of a Class



Task One

- Write a Java class to represent an Employee.
  - An Employee has a name, ID, work experience and basic salary.
  - Write a constructor to add a new employee.
  - Add methods to perform following activities.
    - The gross salary is calculated based on the work experience (WE) as mentioned follows:

      Gross salary = basic salary + Allowance
      - WE >= 1 – gets 20% Allowance
      - WE >= 3 – gets 30% Allowance
      - WE >= 5 – gets 40% Allowance
    - When an employee is promoted, the basic salary is changed. There should be a way to change the WE.
    - There should be a way to view employee details.

Classes in Java

```
class classname
{
  //variable declaration

  //method declaration
}
```

Instance variables/
Non static fields

Class variables/
static fields

Variables in a Class

- **Non-static field (instance variable)** belongs to an individual object.

- **Static field (class variable)**

  is shared by all objects of the class.

Accessing Instance Variables

- An instance variable is called in a particular object using “dot notation”:

  objName.instanceVariable;

```
rect1.color = “YELLOW”;
```

Accessing Class Variables

- Public static fields are referred to in other classes using “dot notation”:

```
ClassName.staticVariable
```

```
double area = Math.PI*r*r;
```

Class Variables

- Class variables can be used to collect statistics or totals for all objects of the class (eg: total number of all bank Accounts).

```
class BankAccount
{
   static int totNumAcc = 0;
...
```

- Static fields can be initialized when declared.

Task Two

- Update the employee class by adding another property, so we can keep track of the number of employees in the company.

Classes in Java

```
class classname
{
 //variable declaration

 //method declaration
}
```

Non static Methods

Static Methods

Calling Non Static Methods

- A non-static method is called for a particular object using “dot notation”:

```
objName.nonStaticMethod(...);
```

```
rect1.setData(10,5,“red”);
```

- Non-static methods can access all fields and call all methods of their class — both static and non-static.

Calling Static Methods

- Static methods can access and manipulate a class's static fields.
- Static methods cannot access non-static fields or call non-static methods of the class.
- Static methods are called using "dot notation":

```
ClassName.staticMethod(...)
```

```
Math.random();
System.exit();
```

Task Three

- Write a method to get the number of employees.
- Write a driver class called EmployeeTest, which will create three employee objects, and print how many employees in the company.

Object Oriented Concepts:

- Abstraction
- Encapsulation
- Polymorphism
- Inheritance

A Class:Abstraction

- Through the process of **abstraction**, a programmer hides all but the relevant data about an object in order to reduce complexity and increase efficiency.
- A class is a general, abstract representation of an object, that specifies the fields and methods that such an object has.
- That object remains as a representation of the original, with unwanted detail omitted.

A Class: Abstraction Example

- For example, we created a Car class that describes the features of all cars (each car has a model, a colour, it can move, etc.).
- The class call **Car** serves as an abstract model for the concept of a car.

Encapsulation

- **Encapsulation** is the mechanism that binds together methods and the data it manipulates, and keeps both safe from outside interference and misuse.
- Encapsulation achieved through by making the fields in a class private and providing access to the fields via public methods.

Scope & Visibility Rules Table

	Scope	Visibility
public variables, methods & classes	public	Visible to all classes
default variables, methods & classes	treated as public within its package	Visible to all classes within the package
private variables, methods & classes	private	Visible only within the class
protected variables, methods & classes	protected	Visible to all classes within the package & inherited classes to outside the package

Method Overloading: Polymorphism

- **Polymorphism** allows different objects to respond to the same message in different ways.
- Polymorphism is achieved through Method Overloading and Method Overriding.

What is Method Overloading?

- Methods of the same class or subclasses that have the **same name** but **different numbers or types of parameters** are called overloaded methods. Overloaded methods may or may not have different return types.

- Use overloaded methods when they perform similar tasks:

```
void setData(int l,int w,String c) {... }
void setData(int l, int w) { ... }
void setData(int l, String c) { ... }
```

Example 1

```
class Rectangle {
	int length;
	int width;
	String color;

	void setData(int l, int w, String c) {
		length = l;
		width = w;
		color = c;
	}
	void setData(int l, int w) {
		length = l;
		width = w;
	}
	void setData(int l, String c) {
		length = l;
		color = c;
	}
	. . .
```

Overloaded Methods

- The compiler treats overloaded methods as completely different methods.
- The compiler knows which one to call based on the number and the types of the parameters passed to the method.

Example 1 contd…

```
class RectTest {
   public static void main(String arg[]){

      Rectangle rect1 = new Rectangle();
      rect1.setData(15,10,“red”);
      rect1.setData(100,50);
   }
}
```

Task Four

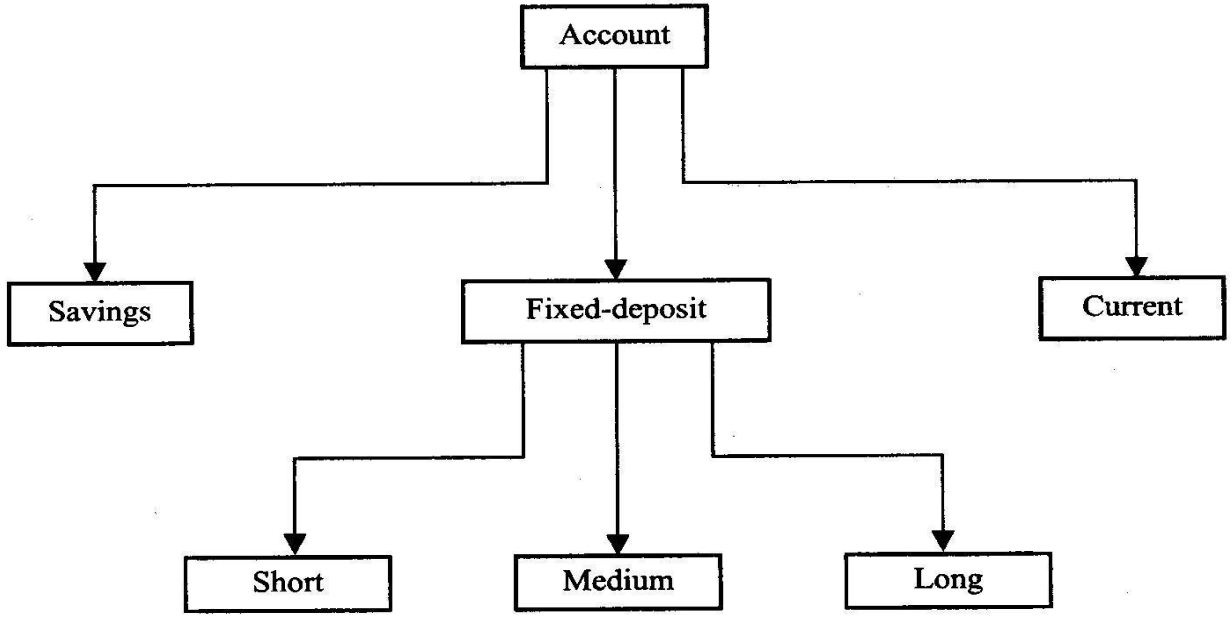
- Bonus of the employees is calculated in different ways.
  - Usually bonus is calculated as 10% of the basic salary.
  - Sometimes a special bonus is calculated based on a rate entered (20%,30% etc)
  - Additionally bonus can also be calculated based on over time (OT). Then the OT hours and the employee class must be entered by the user.
    - Class A employees rate is 30%
    - Class B employees rate is 20%
    - Class C employees rate is 10%
    - Bonus = (Basic salary * rate)*OT hours
      (12000*0.1) * 5 = 6000
  - Write all the overloaded methods called **calcBonus** to the Employee class

Inheritance

- Inheritance can be defined as the process where one object acquires the properties of another.
- With the use of inheritance the information is made manageable in a hierarchical order.

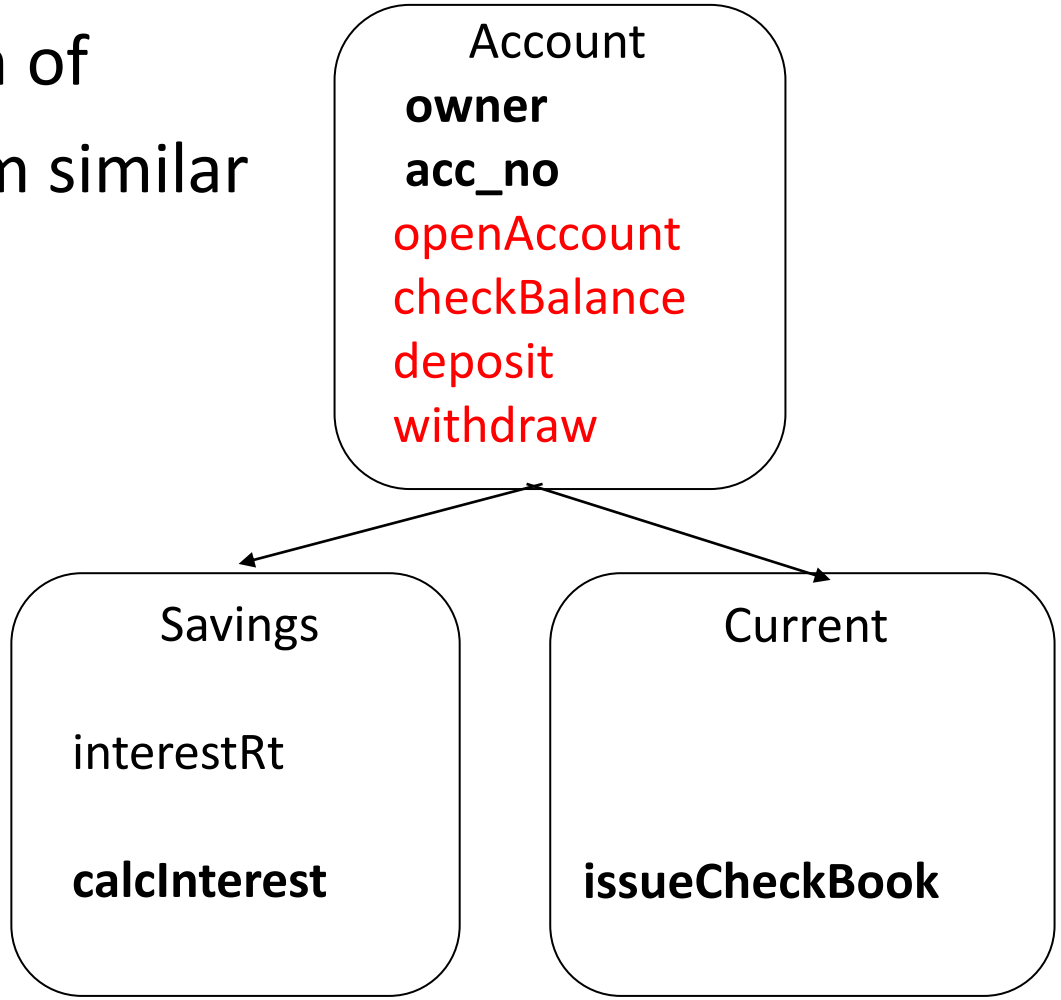
Example

- Below is a hierarchy where certain features of one level are shared by many others below the level.

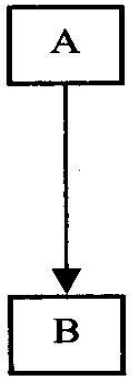


Example

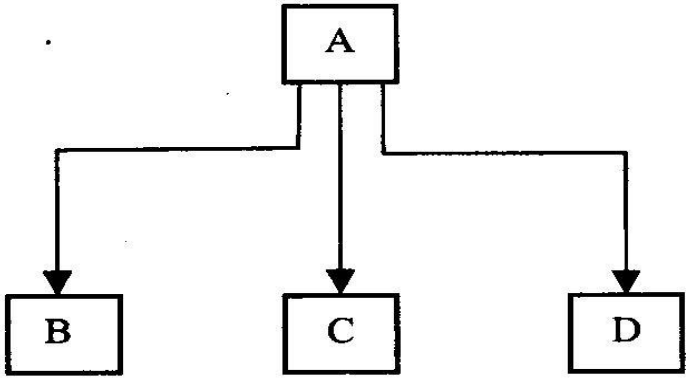
- Inheritance helps to reduce duplication of code by factoring out common code from similar classes into a common super class.



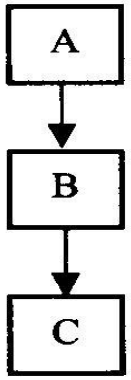
Different forms of Inheritance



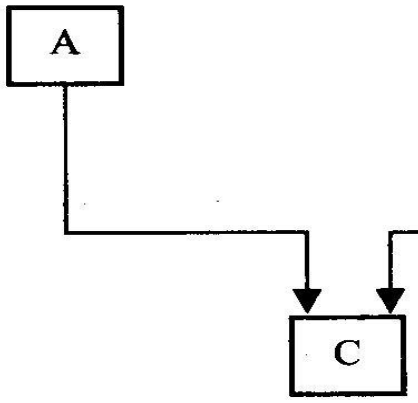
(a) Single inheritance



(b) Hierarchical inheritance



(c) Multilevel inheritance



(d) Multiple inheritance

Java does not directly implement Multiple Inheritance

Single Inheritance

```
class subclassname extends superclassname
{
      //variables declaration

      //methods declaration
}
```

Example: Rectangle class

```
class Rectangle {
      int length;
      int width;

      Rectangle(int x, int y) {
            length = x;
            width = y;
      }
      int area( ) {
            return (length * width);
      }
}
```

Example: Cube class

```
class Cube extends Rectangle {

      int height;

      Cube(int x, int y, int z) {
            super(x, y);
            height = z;
      }
      int volume( ) {
            return (length * width * height);
      }
}
```

Example: Driver class/Main class

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

       Cube c1 = new Cube(14, 12, 10);
       int a = c1.area( );
       int v = c1.volume( );

       System.out.println ("Area = "+ a);
       System.out.println ("Volume = "+ v) ;
   }
}
```

Calling Superclass's Constructor

- **super** may be used within a subclass constructor.

```
Cube(int x, int y, int z){
      super (x, y) ;
      ...
 }
```

Calls Rectangle's constructor

- The number / types of parameters passed to **super** must match parameters of one of the superclass's constructors.
- If present, must be the first statement.

Calling Superclass's Methods

- **super** can be used within a subclass method.

```
int volume( )
 {
     return (super.area( ) * height);
 }
```

Calls Rectangle's area ()

- super.a() refers to method a() in the nearest class, up the inheritance path, where a() is defined.

Overriding Methods

- If a subclass object wants to respond to a method in the parent class with a different behaviour:

  "We should **override it in the subclass,** the method that was defined in the superclass."

How to Override a Method

- Define a method in the subclass having:
  - the same name
  - same arguments
  - same return type

  as a method in the superclass.

- At the method call, the method defined in the subclass is invoked and executed instead of the one in the superclass.

Example

```
class Rectangle {
	int length, width;

	Rectangle(int x, int y) {
		length = x;
		width  = y;
	}
	int area() {
		return (length * width);
	}
}
```

```
class Cube extends Rectangle {
	int height;

	Cube(int x, int y, int z) {
		length = x;
		width = y;
		height = z;
	}

	int area()
	{
		return (length * width * height);
	}
}
```

Example: Driver class

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

        Cube c1 = new Cube(14, 12, 10);
        int a = c1.area();

        System.out.println ("Area = "+ a);
   }
}
```

Task Five

- Create the following classes covering the OO concepts you have learned.
- There are two kinds of employees. Part time employees and Full time employees. (inheritance)
- For part time employees you have to store the hourly rate. All full time employees have an EPF number (eg: 45) and EPF rate. (8%)

Task Five

- Write constructers and following methods.
- Part time employees the salary is calculated as follows. (method overloading)
  - Salary = hourly rate * number on hours worked.
  - The number of hours is entered by the user.
- When you print the full time employee details you have to print EPF number as well. (method overriding)
- Salary of the Full Time employee is calculated as follows. (method overriding)
  - Gross salary = Basic salary - (EPF rate) + Allowance

Final Variables, Methods and Classes

- Final Variable - the value of a particular variable can never be changed.
- Final Method - Functionality defined in this method will never be altered in any way. (no method overriding)
- Final class - the class that cannot be sub classed.

Abstract Methods and Classes

- Abstract Classes
  - Classes that cannot be used to create objects (instantiated).
  - They need to be extended by a sub class.

- Abstract Methods
  - Methods that always have to be implemented in an eventual subclass.

More about Abstract

```
abstract class Shape
{
    ……
    ……
    abstract void draw( );
    ……
    ……
}
```

- When a class contains one or more abstract methods, it should also be declared abstract.
- We cannot declare abstract constructors or abstract static methods

Summery

- Through the process of abstraction, a programmer hides all but the relevant data about an object in order to reduce complexity and increase efficiency.
- Encapsulation is the mechanism that binds together methods and the data it manipulates and keeps both safe from outside interference and misuse.
- Polymorphism allows different objects to respond to the same message in different ways.
- Inheritance can be defined as the process where one object acquires the properties of another.

Thank you

ROBERT GORDON UNIVERSITY ABERDEEN
TEF Gold
INFORMATICS INSTITUTE OF TECHNOLOGY