COMP2511

Object Oriented Programming (OOP) in Java

Prepared by

Dr. Ashesh Mahidadia

OOP in Java

- Object Oriented Programming (OOP)
- Inheritance in OOP
- Introduction to Classes and Objects
- Subclasses and Inheritance
- Abstract Classes
- Single Inheritance versus Multiple Inheritance
- Interfaces
- Method Forwarding (Has-a relationship)
- Method Overriding (Polymorphism)
- Method Overloading
- Constructors

Object Oriented Programming (OOP)

In procedural programming languages (like ‘C’), programming tends to be **action-oriented**,

whereas in Java - programming is **object-oriented**.

In **procedural** programming,

- groups of actions that perform some task are formed into functions and functions are grouped to form programs.

In **OOP**,

- programmers concentrate on creating their own user-defined types called classes.
- each class contains data as well as the set of methods (procedures) that manipulate the data.
- an instance of a user-defined type (i.e. a class) is called an object.
- OOP encapsulates data (attributes) and methods (behaviours) into objects, the data and methods of an object are intimately tied together.
- Objects have the property of *information hiding*.

Inheritance in Object Oriented Programming (OOP)

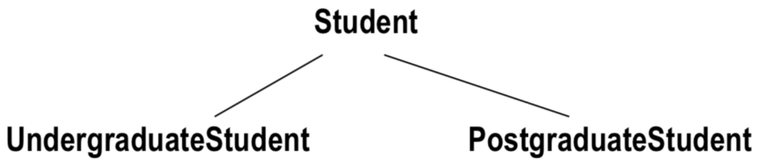
- **Inheritance** is a form of software reusability in which new classes are created from the existing classes by absorbing their attributes and behaviours.

- Instead of defining completely (separate) new class, the programmer can designate that the new class is to **inherit** attributes and behaviours of the existing class (called **superclass**). The new class is referred to as **subclass**.

- Programmer can add more attributes and behaviours to the *subclass*, hence, normally subclasses have more features than their *super classes*.

Inheritance in Object Oriented Programming (OOP)

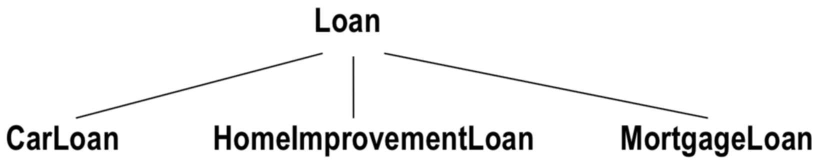
Inheritance relationships form tree-like hierarchical structures. For example,



Student

UndergraduateStudent

PostgraduateStudent

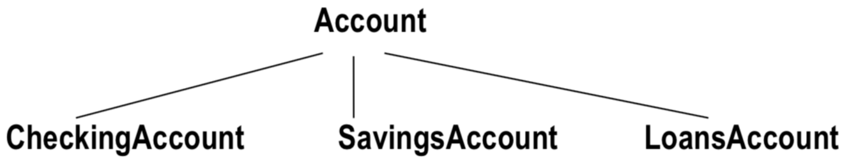


Loan

CarLoan

HomeImprovementLoan

MortgageLoan

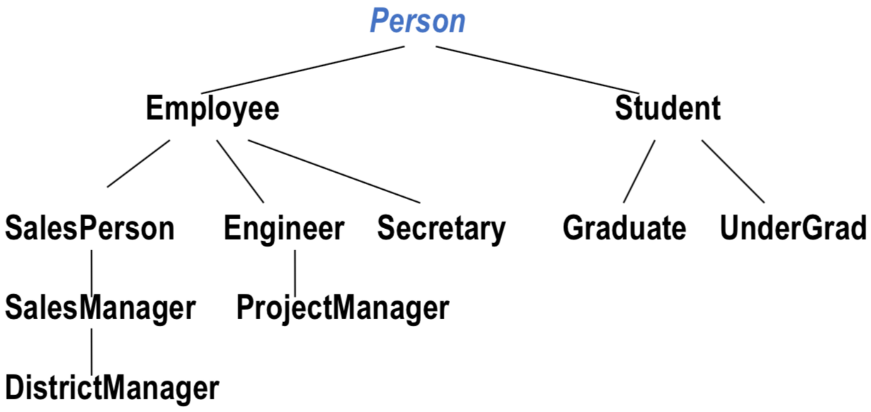


Account

CheckingAccount

SavingsAccount

LoansAccount



Person

Employee

Student

SalesPerson

Engineer

Secretary

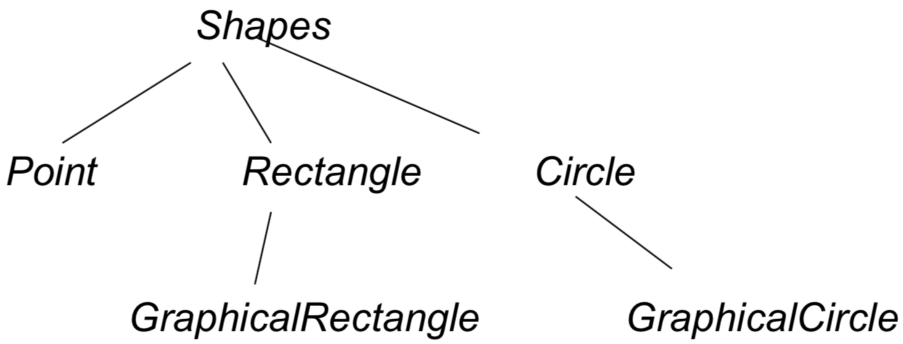
Graduate

UnderGrad

SalesManager

ProjectManager

DistrictManager



Shapes

Point

Rectangle

Circle

GraphicalRectangle

GraphicalCircle

“*Is-a*” - Inheritance relationship

- In an “**is-a**” relationship, an object of a subclass may also be treated as an object of the superclass.

- For example, *UndergraduateStudent* can be treated as *Student* too.

- You should use *inheritance* to model “is-a” relationship.

Very Important:

- Don’t use inheritance unless **all or most** inherited attributes and methods make sense.
- For example, mathematically a *circle* is-a (an) *oval*, however you should **not** inherit a class *circle* from a class *oval*. A class *oval* can have one method to set *width* and another to set *height*.

"*Has-a*" - Association relationship

- In a "has-a" relationship, a **class object has an object of another class** to store its state or do its work, i.e. it "has-a" reference to that other object.
- For example, a Rectangle Is-NOT-a Line.
  However, we may use a Line to draw a Rectangle.
- The "has-a" relationship is quite different from an "is-a" relationship.
- "Has-a" relationships are examples of creating new classes by *composition* of existing classes (as oppose to extending classes).

Very Important:

- Getting "Is-a" versus "Has-a" relationships correct is both subtle and potentially critical. You should consider all possible future usages of the classes before finalising the hierarchy.
- It is possible that obvious solutions may not work for some applications.

Designing a Class

- Think carefully about the functionality (methods) a class should offer.
- Always try to keep data private (local).
- Consider different ways an object may be created.
- Creating an object may require different actions such as initializations.
- Always initialize data.
- If the object is no longer in use, free up all the associated resources.
- Break up classes with too many responsibilities.
- In OO, classes are often closely related. “Factor out” common attributes and behaviours and place these in a class. Then use suitable relationships between classes (for example, “is-a” or “has-a”).

Introduction to Classes and Objects

- A class is a collection of data and methods (procedures) that operate on that data.

- For example,

  a **circle** can be described by the x, y position of its centre and by its radius.

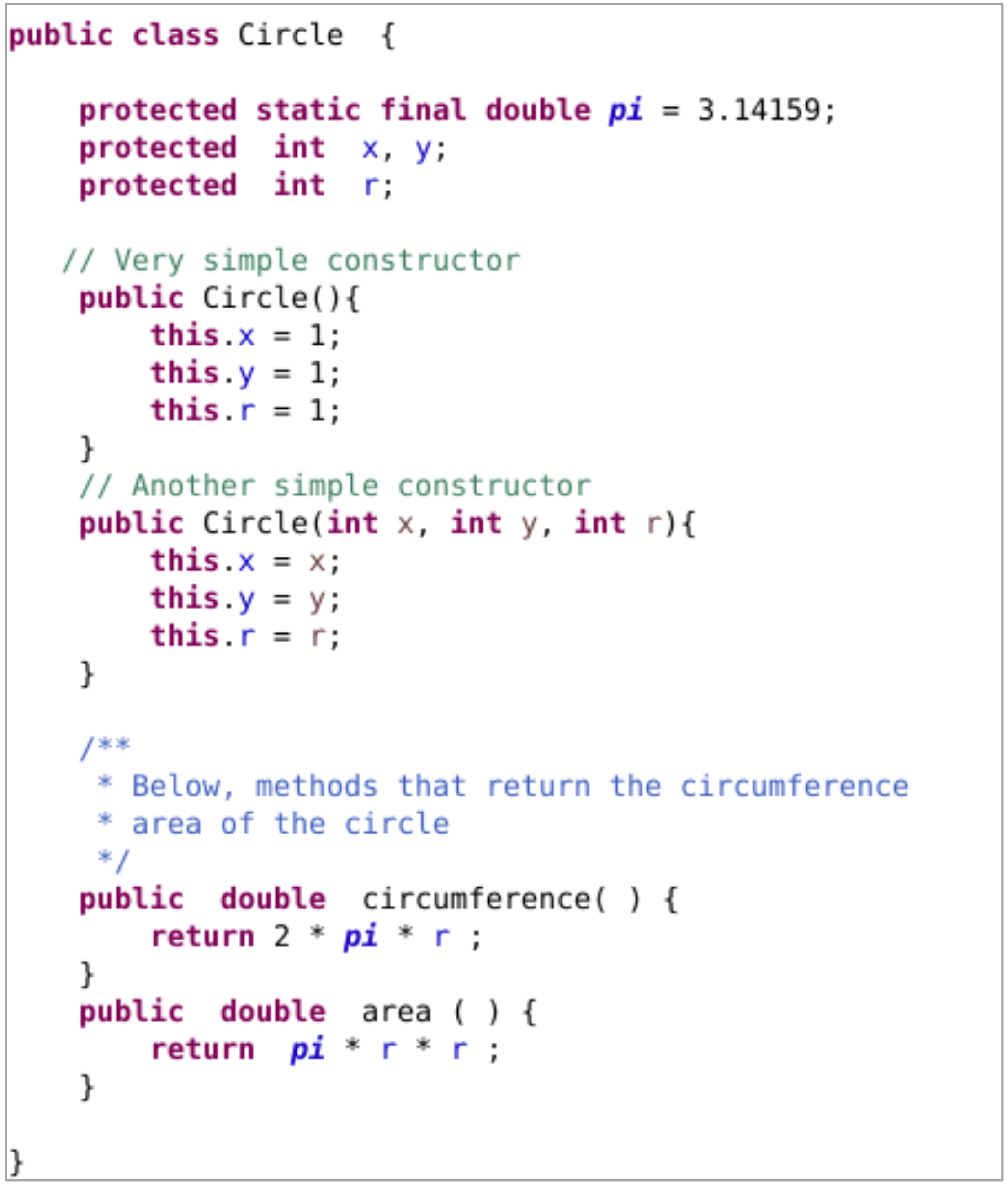
- We can define some useful methods (procedures) for circles,

  compute circumference, compute area, check whether pointes are inside the circle, etc.

- By defining the **Circle** class (as below), we can create a new data type.

The class Circle

- For simplicity, the methods for *getter* and *setters* are not shown in the code.



```java
public class Circle  {

  protected static final double pi = 3.14159;
  protected  int  x, y;
  protected  int  r;

 // Very simple constructor
  public Circle(){
      this.x = 1;
      this.y = 1;
      this.r = 1;
  }
  // Another simple constructor
  public Circle(int x, int y, int r){
      this.x = x;
      this.y = y;
      this.r = r;
  }

  /**
   * Below, methods that return the circumference
   * area of the circle
   */
  public  double  circumference( ) {
      return 2 * pi * r ;
  }
  public  double  area ( ) {
      return  pi * r * r ;
  }

}
```

Objects are Instances of a class

In Java, objects are created by instantiating a class.

For example,

```
Circle  c ;
c =  new  Circle ( ) ;
```

OR

```
Circle  c  = new  Circle ( ) ;
```

Accessing Object Data

We can access data fields of an object.

For example,

```
Circle c = new Circle ( ) ;

// Initialize our circle to have centre (2, 5)
// and radius  1.
// Assuming, x, y and r are not private

c.x = 2;
c.y = 5;
c.r = 1;
```

Using Object Methods

To access the methods of an object, we can use the same syntax as accessing the data of an object:

```
Circle  c = new  Circle ( ) ;
double  a;

c.r = 2;      // assuming r is not private

a  =  c.area( );

//Note that its not :   a = area(c)  ;
```