**Abstract Classes and Methods**

We have seen the class hierarchies - Entertainments example in Java Class Inheritance and IceCreamPrices example in Inheritance Example Program To Remove Duplicate Code. The class hierarchies look similar except that there is minor difference between them. IceCream is a concrete super-class where as Entertainment is an abstract super-class. The difference is explained below.

In reality IceCream can exist, even with out FruitSaladWithIceCream, where as Entertainment can not exist on its own. It has to be either Movie, Drama or Circus or some other entertainment. Similarly Metal is an abstract class, since for we to show the Metal, we need to either show Gold or Iron or some thing else. If we had Mobile and MobileWithCamera, then Mobile would be a concrete class since it can exists on its own.

IceCream and Mobile are examples of concrete super-classes, where as Entertainment, Metal and Shape are examples of abstract super-classes.

In Java every class defined is by default concrete class, but if we want to implement abstract class, we need to use the abstract keyword before the class to mark a class as abstract class.

abstract class Shape  
{  
}  
  
class Rectangle extends Shape  
{  
}

Here we have created an *abstract* class called Shape and a *concrete* class called Rectangle. Abstract class is prefixed with the keyword abstract, where as for concrete class we need not mention any keyword.

Shape shape1; // Can create a reference of abstract class  
Shape shape2 = new Shape(); // WILL NOT WORK. Can not create an object of abstract class  
  
Rectangle rect1 = new Rectangle(); // Can create reference and object of a concrete class  
shape1 = rect1; // Can assign concrete sub-class reference to abstract super-class reference

* We can only create a reference for an abstract class, but can not create (or instantiate) an object of abstract class.
* We can create both reference and object for any concrete class.
* We can assign a concrete sub-class reference to abstract super-class reference. i.e. rect1 can be assigned to shape1.

The other advantage of abstract classes is we can define abstract methods.

**Abstract Methods**

An abstract classes do not map to real life objects and they can not exist on their own, with out taking the form any of its concrete sub-classes. e.g., Shape can not exist on it own, it has to be either Rectangle or Square or Circle.

Abstract methods are those methods which have only the *declaration* but do not have a *definition*. Declaration means creating only the method signature (the method name, parameters and return type), but no method body, where as definition means creating the method signature and the method body as well.

For e.g., we know that every Shape has an area, but we do not know how to calculate the area, until we know what Shape it is. This is because the area calculation for Rectangle is different from the area calculation of Triangle, which is different from that of the Circle. So we will declare that we have a method called getArea() in Shape, but only define or implement the logic of calculating the area in there respective concrete sub-classes.

class CalculateAreas  
{  
    public static void main(String arg[])  
    {  
        Rectangle rect = new Rectangle(5.25, 4.0);  
        System.out.println("Area of rectangle is " + rect.getArea());  
          
        Circle circle = new Circle(7.5);  
        System.out.println("Area of circle is " + circle.getArea());      
    }  
}  
  
abstract class Shape  
{  
    abstract double getArea(); // LINE A  
}  
  
class Rectangle extends Shape  
{  
    double length;  
    double breadth;  
  
    Rectangle(double length, double breadth)  
    {  
        this.length = length;  
        this.breadth = breadth;  
    }  
  
    double getArea()  
    {  
        return length \* breadth;  
    }  
}  
  
class Circle extends Shape  
{  
    double radius;  
  
    Circle(double radius)  
    {  
        this.radius = radius;  
    }  
  
    double getArea()  
    {  
        return 3.14 \* radius \* radius;  
    }  
}

**Important**

Here we have created an abstract Shape class and two concrete classes namely Rectangle and Circle which extends from the Shape class. We have implemented the getArea() method in the sub-classes. **Please note that every sub-class extending from an abstract class should implement all the abstract methods, otherwise it will cause compilation errors.**

**Rules for abstract method and abstract classes**

* A class can be marked as abstract with out containing any abstract method. But if a class has even one abstract method, then the class has to be an abstract class.

abstract class A  
{  
    // Valid, even with out any abstract methods  
}  
  
class B // Invalid, class B should be abstract, since it has abstract method.  
{  
    abstract void method1();  
}

* An abstract class can have one or more abstract methods.

abstract class C  
{  
    abstract void method1();  
  
    abstract double method2(int x, int y);  
  
    abstract boolean method3(char z);  
}

* An abstract class can have both abstract and non abstract (or concrete) method.

abstract class D  
{  
    void method1()  
    {  
        System.out.println("I am a concrete method");  
    }  
  
    abstract double method2(int x, int y);  
  
    int method3(double z)  
    {  
        System.out.println("I am also a concrete method");  
return 0;  
    }  
  
    abstract boolean method4(char z);  
}

* The abstract method should not have method body. Even empty flower braces { } are not allowed.

abstract class A  
{  
    abstract void method1(); // Valid  
  
    abstract void method2() {} // Invalid - since it has method body  
  
}

* Any sub-class extending from an abstract class should either implement all the abstract methods of the super-class or the sub-class itself should be marked as abstract.

abstract class A  
{  
    abstract void method1();  
  
    abstract void method2();  
  
}  
  
class B extends A  
{  
    // Invalid since B does not implement the abstract methods  
}  
  
abstract class C extends A  
{  
    // Valid since C is marked as abstract, even though the abstract methods are not implemented,  
}  
  
class D extends A  
{  
    void method1()  
    {  
        System.out.println("Method1 implemented here.");  
    }  
  
    // Invalid, class D should be marked as abstract, since method2 is not implemented.  
}  
  
abstract class E extends A  
{  
    void method1()  
    {  
        System.out.println("Method1 implemented here.");  
    }  
  
    // Even though method2 is not implemented, class D is marked as abstract, so it is Valid.  
}  
  
  
class F extends A  
{  
    // Valid since both methods are implemented here.  
    void method1()  
    {  
        System.out.println("Method1 implemented here.");  
    }  
  
    void method2()  
    {  
        System.out.println("Method2 implemented here.");  
    }  
}

* If an abstract class contains multiple methods, it is not necessary that all the methods of the abstract class are implemented in the immediate sub-class. Few of them can be implemented in sub-sub-classes or any where else in the sub-class hierarchy. But for a class to be concrete, all the abstract methods in its super-class must be implemented.

abstract class X  
{  
    abstract void method1();  
    abstract void method2();  
}  
  
abstract class Y extends X  
{  
    void method1()  
    {  
        System.out.println("Method1 implemented here.");  
    }  
}  
  
class Z extends Y  
{  
    void method2()  
    {  
        System.out.println("Method2 implemented here.");  
    }  
}

* It is not necessary to add the abstract methods only in the super most class, we can add more abstract methods in the sub-classes.

abstract class X  
{  
    abstract void method1();  
}  
  
abstract class Y extends X  
{  
    abstract void method2();  
}  
  
class Z extends Y  
{  
    void method1()  
    {  
        System.out.println("Method1 from class X implemented here.");  
    }  
  
    void method2()  
    {  
        System.out.println("Method1 from class Y implemented here.");  
    }  
}

**Creating Array Of Objects in Java**

We can also create array of objects references and initialize data for each reference (or element) of the array.

class Student  
{  
    String name;  
    int marks;  
    char section;  
}

Student[] students = new Student[3];

**Please note that, when we create an array, only the references are created and not the objects. Hence we need to create the objects separately**

students[0] = new Student(); // LINE A  
students[1] = new Student();

students[2] = new Student();

**Java Program to find Largest Area By comparing various shapes**

class FindShapeWithLargestArea  
{  
    public static void main(String arg[])  
    {  
        Shape shapes[] = new Shape[5];  
          
        shapes[0] = new Rectangle(30.3, 45.4);  
        shapes[1] = new Circle(23.2);  
        shapes[2] = new Triangle(146.2, 40.0);  
        shapes[3] = new Rectangle(25, 57.8);  
        shapes[4] = new Triangle(120.0, 8.0);  
          
          
        Shape largest = shapes[0];  
          
        for(int i = 0; i < shapes.length; i++)  
        {  
            if(shapes[i].getArea() > largest.getArea())  
            {  
                largest = shapes[i];  
            }  
        }  
          
        System.out.println("The details of the shape with largest area are :" );  
          
        largest.printDetails();      
    }  
}  
  
abstract class Shape  
{  
    abstract double getArea();  
  
    abstract void printDetails();  
}  
  
class Rectangle extends Shape  
{  
    double length;  
    double breadth;  
  
    Rectangle(double length, double breadth)  
    {  
        this.length = length;  
        this.breadth = breadth;  
    }  
  
    double getArea()  
    {  
        return length \* breadth;  
    }  
  
    void printDetails()  
    {  
        System.out.println("Type = Rectangle");  
        System.out.println("Length = " + length);  
        System.out.println("Breadth = " + breadth);  
        System.out.println("Area = " + getArea());  
    }  
}  
  
class Circle extends Shape  
{  
    double radius;  
  
    Circle(double radius)  
    {  
        this.radius = radius;  
    }  
  
    double getArea()  
    {  
        return 3.14 \* radius \* radius;  
    }  
  
    void printDetails()  
    {  
        System.out.println("Type = Circle");  
        System.out.println("Radius = " + radius);  
        System.out.println("Area = " + getArea());  
    }  
}  
  
class Triangle extends Shape  
{  
    double base;  
    double height;  
  
    Triangle(double base, double height)  
    {  
        this.base = base;  
        this.height = height;  
    }  
  
    double getArea()  
    {  
        return base \* height / 2.0;  
    }  
  
    void printDetails()  
    {  
        System.out.println("Type = Triangle");  
        System.out.println("Base = " + base);  
        System.out.println("Height = " + height);  
        System.out.println("Area = " + getArea());  
    }  
}