Abstract Class and Interface

Example -1

An abstract class is a class that you cannot create an instance of. It can provide basic functionality, but in order for that functionality to be used, one or more other classes must derive from the abstract class.

One of the major benefits of abstract classes is that you can reuse code without having to retype it. That has a plethora of benefits, such as reducing bugs and making coding faster.

A concrete example of an abstract class would be a class called Animal. You see many animals in real life, but there are only kinds of animals. That is, you never look at something purple and furry and say "that is an animal and there is no more specific way of defining it".

Instead, you see a dog or a cat or a pig... all animals. The point is, that you can never see an animal walking around that isn't more specifically something else (duck, pig, etc.).

The Animal is the abstract class and Duck/Pig/Cat are all classes that derive from that base class. Animals might provide a function called "Age" that adds 1 year of life to the animals. It might also provide an abstract method called "IsDead" that, when called, will tell you if the animal has died. Since IsDead is abstract, each animal must implement it. So, a Cat might decide it is dead after it reaches 14 years of age, but a Duck might decide it dies after 5 years of age.

The abstract class Animal provides the Age function to all classes that derive from it, but each of those classes has to implement IsDead on their own.

Now, an interface is like an abstract class, except it does not contain any logic. Rather, it specifies an interface. So, there might be an interface called IFly. This might have the methods GoForward and GoDown. Those methods would not actually contain any logic... each class that implements interface IFly would have to implement those GoForward and GoDown methods. You could have classes Duck and Finch implement interface IFly. Then, if you want to keep a list of instances that can fly, you just create a list that contains items of type IFly. That way, you can add Ducks and Finches and any other instance of a class the implements IFly to the list.

So, abstract classes can be used to consolidate and share functionality, while interfaces can be used to specify what the common functionality that will be shared between different instances will be, without actually building that functionality for them. Both can help you make your code smaller, just in different ways. There are other differences between interfaces and abstract classes, but those depend on the programming language, so I won't go into those other differences here.

Example -2

Do you mean real-world as in "A live software system which includes  Abstract classes or interfaces" or do you mean "A contrived example which demonstrates their usefullness"?

If you mean the latter think of Vehicle as an abstract class. You can't yet do anything with it because you have no idea what it does, or how to drive it.

abstract class Vehicle{}

Vehicles could be split into morotized and pedal-powered, but still this is abstract, we still dont know what to do with it.

abstract class MotorVehicle : Vehicle {}

abstract class PedaledVehicle: Vehicle {}

You could now define a concrete (non-abstract) class, like car.

class MotorCar : MotorVehicle {}

 Intefaces come in handy you can only inherit from one base class. So imagine some vehicles are drivable, others are remote controlled, some vehicles use a stearing wheel, others don’t

interface IDrivable{}

interface IHasStearingWheel{}

Now you could derive a DrivableMotorCar from its base clas, and also implement other behaviours.

class DrivableMotorCar : MotorVehicle, IDrivable, IHasStearingWheel {}

Example -3

Write a program to maintain University marking Database.  
**Student** is **Abstract class**, it has Roll no., Name, subject\_1\_mark attributes.  
Show\_student\_data() is abstract method.  
Get\_student\_data() is non abstract method.  
**ISport** is an **Interface**, having attribute sport\_grace\_marks=5.  
Show\_sport\_mark() is a method.  
**IExService\_Man** is an **Interface**, having attribute ExService\_Man\_grace\_marks=10.  
Show\_ExService\_Man\_mark() is a method.  
Result is Class, it is inherited from Student, ISport, IExService\_Man.   
Total\_marks=subject\_1\_mark + sport\_grace\_marks + ExService\_Man\_grace\_marks.  
Show\_result() is method of Result class.

Program

interface IMario

{

Void behavior();

}

class  Mario implements IMario

{

public void behavior()

{

System.out.println("This is simple Mario");

}

}

class  SuperMario implements IMario

{

public void behavior()

{

System.out.println("This is Super Mario with power");

}

}

class  TestMario

{

public static void main(String args[])

{

System.out.println("----game started----");

Imario obj=new Mario();

Obj.behavior();

System.out.println(-----After Mashroom-----");

obj=new SuperMario();

Obj.behavior();

System.out.println(-----After Power Loss-----");

obj=new Mario();

Obj.behavior();

}

}

Program

// Using abstract methods and classes.   
abstract class Figure {   
double dim1;   
double dim2;   
Figure(double a, double b) {   
dim1 = a;   
dim2 = b;   
}   
// area is now an abstract method   
abstract double area();   
}

class Rectangle extends Figure {   
Rectangle(double a, double b) {   
super(a, b);   
}   
// override area for rectangle   
double area() {   
System.out.println("Inside Area for Rectangle.");   
return dim1 \* dim2;   
}   
}

class Triangle extends Figure {   
Triangle(double a, double b) {   
super(a, b);   
}   
// override area for right triangle   
double area() {   
System.out.println("Inside Area for Triangle.");   
return dim1 \* dim2 / 2;   
}   
}

class AbstractAreas {   
public static void main(String args[]) {   
// Figure f = new Figure(10, 10); // illegal now   
Rectangle r = new Rectangle(9, 5);   
Triangle t = new Triangle(10, 8);   
Figure figref; // this is OK, no object is created   
figref = r;   
System.out.println("Area is " + figref.area());   
figref = t;   
System.out.println("Area is " + figref.area());   
}   
}

Program

Person.java

|  |
| --- |
| package com.journaldev.design;    //abstract class  public abstract class Person {        private String name;      private String gender;        public Person(String nm, String gen){          this.name=nm;          this.gender=gen;      }        //abstract method      public abstract void work();        @Override      public String toString(){          return "Name="+this.name+"::Gender="+this.gender;      }    } |

|  |
| --- |
| package com.journaldev.design;    public class Employee extends Person {        private int empId;        public Employee(String nm, String gen, int id) {          super(nm, gen);          this.empId=id;      }        @Override      public void work() {          if(empId == 0){              System.out.println("Not working");          }else{              System.out.println("Working as employee!!");          }      }        public static void main(String args[]){          //coding in terms of abstract classes          Person student = new Employee("Dove","Female",0);          Person employee = new Employee("Pankaj","Male",123);          student.work();          employee.work();          //using method implemented in abstract class - inheritance          employee.changeName("Pankaj Kumar");          System.out.println(employee.toString());      }    } |

runchifyExam.java

Java

|  |
| --- |
| package com.crunchify.tutorial;    import java.util.Date;    /\*\*  \* @author Crunchify.com  \*  \*/    public abstract class CrunchifyExam {    public enum ExamStatus {  PASSED, FAILED  }    private Date examTime;  private ExamStatus status;    public CrunchifyExam(Date examTime, ExamStatus status) {  this.examTime = examTime;  this.status = status;  }    public void getExamTime() {  System.out.println("This is your Exam Status: " + examTime);  }    public void setExamTime(Date examTime) {  this.examTime = examTime;  }    public void setExamStatus(ExamStatus status) {  this.status = status;  }    public ExamStatus getExamStatus() {  return status;  }    abstract public void checkResult();  } |

Crunchify1stSchoolExamResult.java

Java

|  |
| --- |
| package com.crunchify.tutorial;    import java.util.Date;    /\*\*  \* @author Crunchify.com  \*/    public class Crunchify1stSchoolExamResult extends CrunchifyExam {    public Crunchify1stSchoolExamResult(Date examTime, ExamStatus status) {  super(examTime, status);  // TODO Auto-generated constructor stub  }    @Override  public void checkResult() {  String studentName = "Nancy";  String studentResult = "85%";      // School NO-1 will provide all their formula to find if user is passed or failed.  // After detailed calculation let's say student's grade is "PASSED".  ExamStatus examResult = ExamStatus.PASSED;  setExamStatus(examResult);  System.out.println("Hey.. this is Crunchify's user1 " + studentName + "and here School grade is "  + studentResult + " - " + getExamStatus());    }  } |

unchify2ndSchoolExamResult.java

Java

|  |
| --- |
| package com.crunchify.tutorial;    import java.util.Date;    /\*\*  \* @author Crunchify.com  \*/    public class Crunchify2ndSchoolExamResult extends CrunchifyExam {    public Crunchify2ndSchoolExamResult(Date examTime, ExamStatus status) {  super(examTime, status);  // TODO Auto-generated constructor stub  }    @Override  public void checkResult() {  String studentName = "John";  String studentResult = "45%";    // School NO-2 will provide all their formula to find if user is passed or failed.  // After detailed calculation let's say student's grade is "FAILED".  ExamStatus examResult = ExamStatus.FAILED;  setExamStatus(examResult);  System.out.println("Hey.. this is Crunchify's user1 " + studentName + "and here School grade is "  + studentResult + " - " + getExamStatus());    }  } |

Program

This section provides you an example of the abstract class to create an abstract class just use the abstract keyword before the class keyword, in the class declaration .

/\* File name : Employee.java \*/

public abstract class Employee

{

private String name;

private String address;

private int number;

public Employee(String name, String address, int number)

{

System.out.println("Constructing an Employee");

this.name = name;

this.address = address;

this.number = number;

}

public double computePay()

{

System.out.println("Inside Employee computePay");

return 0.0;

}

public void mailCheck()

{

System.out.println("Mailing a check to " + this.name

+ " " + this.address);

}

public String toString()

{

return name + " " + address + " " + number;

}

public String getName()

{

return name;

}

public String getAddress()

{

return address;

}

public void setAddress(String newAddress)

{

address = newAddress;

}

public int getNumber()

{

return number;

}

}

You can observe that except abstract methods the Employee class is same as normal class in Java. The class is now abstract, but it still has three fields, seven methods, and one constructor.

Now you can try to instantiate the Employee class as shown below:

/\* File name : AbstractDemo.java \*/

public class AbstractDemo

{

public static void main(String [] args)

{

/\* Following is not allowed and would raise error \*/

Employee e = new Employee("George W.", "Houston, TX", 43);

System.out.println("\n Call mailCheck using Employee reference--");

e.mailCheck();

}

}

When you compile the above class, it gives you the following error:

Employee.java:46: Employee is abstract; cannot be instantiated

Employee e = new Employee("George W.", "Houston, TX", 43);

^

1 error

Inheriting the Abstract Class:

We can inherit the properties of Employee class just like concrete class as shown below:

/\* File name : Salary.java \*/

public class Salary extends Employee

{

private double salary; //Annual salary

public Salary(String name, String address, int number, double

salary)

{

super(name, address, number);

setSalary(salary);

}

public void mailCheck()

{

System.out.println("Within mailCheck of Salary class ");

System.out.println("Mailing check to " + getName()

+ " with salary " + salary);

}

public double getSalary()

{

return salary;

}

public void setSalary(double newSalary)

{

if(newSalary >= 0.0)

{

salary = newSalary;

}

}

public double computePay()

{

System.out.println("Computing salary pay for " + getName());

return salary/52;

}

}

Here, you cannot instantiate the Employee class, but you can instantiate the Salary Class, and using this instance you can access the all the three fields and seven methods of Employee class as shown below.

/\* File name : AbstractDemo.java \*/

public class AbstractDemo

{

public static void main(String [] args)

{

Salary s = new Salary("Mohd Mohtashim", "Ambehta, UP", 3, 3600.00);

Employee e = new Salary("John Adams", "Boston, MA", 2, 2400.00);

System.out.println("Call mailCheck using Salary reference --");

s.mailCheck();

System.out.println("\n Call mailCheck using Employee reference--");

e.mailCheck();

}

}

This produces the following result:

Constructing an Employee

Constructing an Employee

Call mailCheck using Salary reference --

Within mailCheck of Salary class

ailing check to Mohd Mohtashim with salary 3600.0

Call mailCheck using Employee reference--

Within mailCheck of Salary class

ailing check to John Adams with salary 2400.

Abstract Methods:

If you want a class to contain a particular method but you want the actual implementation of that method to be determined by child classes, you can declare the method in the parent class as abstract.

abstract keyword is used to declare the method as abstract.

You have to place the abstract keyword before the method name in the method declaration.

An abstract method contains a method signature, but no method body.

Instead of curly braces an abstract method will have a semoi colon ( ; ) at the end.

Below given is an example of the abstract method.

public abstract class Employee

{

private String name;

private String address;

private int number;

public abstract double computePay();

//Remainder of class definition

}

Declaring a method as abstract has two consequences:

The class containing it must be declared as abstract.

Any class inheriting the current class must either override the abstract method or declare itself as abstract.

Note: Eventually, a descendant class has to implement the abstract method; otherwise, you would have a hierarchy of abstract classes that cannot be instantiated.

Suppose Salary class is inherits the Employee class, then it should implement the computePay() method as shown below:

/\* File name : Salary.java \*/

public class Salary extends Employee

{

private double salary; // Annual salary

public double computePay()

{

System.out.println("Computing salary pay for " + getName());

return salary/52;

}

//Remainder of class definition

}