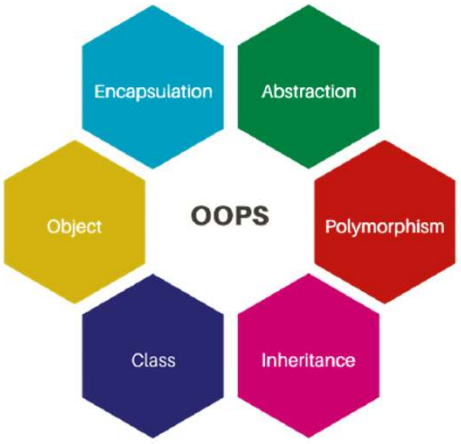


ABESIT COLLEGE OF ENGINEERING
CRC-Training

Java™

Abstract Class

ABESIT COLLEGE OF ENGINEERING
CRC-Training

Why and Where we should use abstract class rather than a normal class

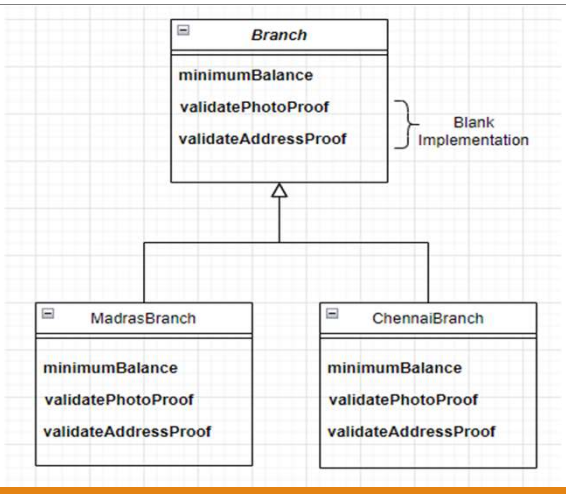
Let see a scenario:

- The bank needs **a photo identification proof, address proof and minimum deposit amount of Rs 1000** from the customer to open his account with the bank.
- Each branch of the bank **has its own process for validating the address proof and photo identification proof** for the customers.
- The open account process validates the photo identification proof, address proof and minimum balance for opening an account.

Why and Where we should use abstract class rather than a normal class

ABESIT COLLEGE OF ENGINEERING
CRC-Training

based on the scenario we can solve by:



Why and Where we should use abstract class rather than a normal class

ABESIT COLLEGE OF ENGINEERING
CRC-Training

- So this is **not a good coding practice** to keep **blank implementation.**
- Because on instantiating Branch class object and if you **call the methods** on it then there will be **no output**.

So we need to find a way to **stop the instantiation of Branch class**.

ABESIT COLLEGE OF ENGINEERING
CRC-Training

Is it mandatory for a subclass to override, always the methods of the superclass?

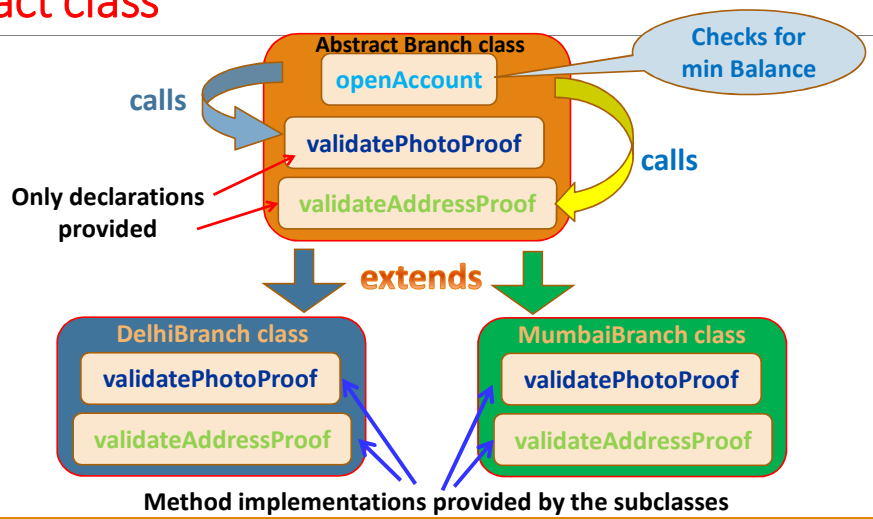
- In our scenario, it is **mandatory** for **all subclass of the branch to override** validatePhotoProof(String proof) and validateAddressProof(String proof) methods and give their own implementation.

So, let us see how to **prevent** **the instantiation of the superclass** and also how will it be **mandatory** for the subclass to **override** the methods from the superclass.

Note – This will achieve through creating a abstract

ABESIT COLLEGE OF ENGINEERING
CRC-Training

abstract class



ABESIT COLLEGE OF ENGINEERING
CRC-Training

Other Example of Abstract Class

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**.

ABESIT COLLEGE OF ENGINEERING
CRC-Training

Abstract Keyword

- Abstract keyword means incomplete.
- The abstract keyword is used to achieve abstraction in Java.
- Abstraction lets you focus on what the object does instead of how it does it.
- There are two ways to achieve abstraction in java
  - Abstract class (0 to 100%)
  - Interface (100%)

ABESIT COLLEGE OF ENGINEERING
CRC-Training

Abstract Keyword

- The abstract keyword is a non-access modifier which is used to create **abstract class** and **method**.
- **Abstarct Class:** An abstract class is a restricted class that cannot be used to create objects (to access it, it must be inherited from another class).
- **Abstarct Method:** An abstract method can only be used in an abstract class, and it does not have a body. The body is provided by the subclass (inherited from).

ABESIT COLLEGE OF ENGINEERING
CRC-Training

Rules of abstract keyword

Don'ts

- An abstract keyword cannot be used with variables and constructors.
- If a class is abstract, it cannot be instantiated by new keyword.
- If a method is abstract, it doesn't contain the body.
- We cannot use the abstract keyword with the **final**.
- We cannot declare abstract methods as **private**.
- We cannot declare abstract methods as **static**.

ABESIT COLLEGE OF ENGINEERING
CRC-Training

Rules of abstract keyword

Do's

- An abstract keyword can only be used with class and method.
- An abstract class can contain constructors and static methods.
- If a class extends the abstract class, it must also implement all of the abstract method.
- An abstract class can contain the main method and the final method.
- An abstract class can contain overloaded abstract methods.

ABESIT COLLEGE OF ENGINEERING
CRC-Training

What is an Abstract Class

- An Abstract class is a **conceptual class.**
- An Abstract class cannot be instantiated – **objects cannot be created by new keyword.**
- Abstract classes provides a common root for a group of classes, nicely tied together in a package.
- When we define a class to be “final”, it cannot be extended. In certain situation, we want properties of classes to be always extended and used. Such classes are called **Abstract Classes**.

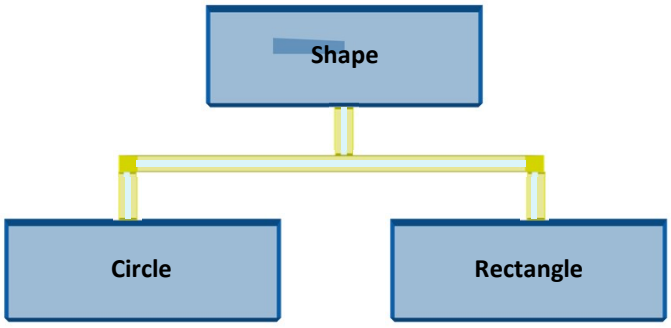
ABESIT COLLEGE OF ENGINEERING
CRC-Training

Properties of an Abstract Class

- A class with one or more abstract methods is automatically abstract and it cannot be instantiated.
- A class declared abstract, even with no abstract methods can not be instantiated.
- A subclass of an abstract class can be instantiated if it overrides all abstract methods by implementing them.
- A subclass that does not implement all of the superclass abstract methods is itself abstract; and it cannot be instantiated.
- We cannot declare abstract constructors or abstract static methods.
- A subclass of a non-abstract superclass can be abstract.

ABESIT COLLEGE OF ENGINEERING
CRC-Training

Abstract Class Example



ABESIT COLLEGE OF ENGINEERING
CRC-Training

Declaration of an Abstract Class

```
abstract class ClassName
{
    ...
    ...
    abstract DataType MethodName1();
    ...
    ...
    DataType Method2()
    {
      // method body
    }
}
```

Example

```
abstract public class Shape
{

    public abstract double area();

    public void move()
     { // non-abstract method
       // implementation
     }
}
```


ABESIT COLLEGE OF ENGINEERING
CRC-Training

Is the following statement valid?

~~Shape sh = new Shape();~~

ABESIT COLLEGE OF ENGINEERING
CRC-Training

Implementation of an Abstract Class

```
public class Circle extends Shape {
        private double r;
        private static final double PI =3.1415926535;
        public Circle() { r = 1.0; }
        public double area() { return PI * r * r; }
 …
 }
public class Rectangle extends Shape {
        private double l, b;
        public Rectangle() { l = 0.0; b=0.0; }
        public double area() { return l * b; }
 ...
 }
```


ABESIT COLLEGE OF ENGINEERING
CRC-Training

Implementation of an Abstract Class

```
1  abstract class Shape
2  {
3          public abstract double area();
4          public void move()
5            { // non-abstract method
6              // implementation
7            }
8  }
9
10 class Circle extends Shape
11 {
12     double r;
13     double PI =3.1415926535;
14     public Circle() { r = 1.0; }
15     public double area(){
16         return PI * r * r;
17     }
18 }
```

```
19
20 class Rectangle extends Shape
21 {
22     private double l, b;
23     public Rectangle() { l = 0.0; b=0.0; }
24     public double area() { return l * b; }
25 }
26
27 public class Main
28 {
29     public static void main(String[] args) {
30     Shape obj1;
31     obj1=new Circle();
32     System.out.println("Circle area: " + obj1.area());
33     obj1=new Rectangle();
34     System.out.println("Rectangle area: " + obj1.area());
35     }
36 }
```

```
Circle area: 3.1415926535
Rectangle area: 0.0
```

ABESIT COLLEGE OF ENGINEERING

CRC-Training

Abstract Class Example

```
abstract class Branch{
  abstract boolean validatePhotoProof(String proof);
  abstract boolean validateAddressProof(String proof);

  void openAccount(String photoProof,String addressProof,int amount){
    if(amount>=1000){
      if(validateAddressProof(addressProof) && validatePhotoProof(photoProof)){
        System.out.println("Account opened");
      }
      else{
        System.out.println("cannot open account");
      }
    }
    else{
      System.out.println("cannot open account");
    }
  }
}
```


ABESIT COLLEGE OF ENGINEERING

CRC-Training

Abstract Class Example

```
class ChennaiBranch extends Branch{
  boolean validatePhotoProof(String proof){
    if(proof.equalsIgnoreCase("pan card")){
      return true;
    }
    return false;
  }
  boolean validateAddressProof(String proof){
    if(proof.equalsIgnoreCase("ration card")){
      return true;
    }
    return false;
  }
}
class Main{
public static void main(String[] args){
  Branch chennaiBranch=new ChennaiBranch();
  chennaiBranch.openAccount("pan card","ration card",2000);
 }
}
```

```
Account opened
```

```
class DelhiBranch extends Branch{
 boolean validatePhotoProof(String proof){
   if(proof.equalsIgnoreCase("pan card")){
     return true;
   }
   return false;
 }
 boolean validateAddressProof(String proof){
   if(proof.equalsIgnoreCase("electric bill")){
     return true;
   }
   return false;
 }
}

 class Main{
 public static void main(String[] args){
   Branch delhiBranch=new DelhiBranch();
   delhiBranch.openAccount("pan card","electric bill",2500);
  }
}
```

```
Account opened
```

ABESIT COLLEGE OF ENGINEERING
CRC-Training

Other example of an Abstract Class

Example:

```
abstract class Account
{
    public void deposit()
    {
    }
    public void withdraw()
    {
    }
    public void checkBalance()
    {
    }
    public abstract float rateOfInterest();
    public abstract String[] services();
}
```

```
class SavingAccount extends Account
{
    public float rateOfInterest()
    {    }
    public String[] services()
    {    }
}

class CurrentAccount extends Account
{
    public float rateOfInterest()
    {    }
//Now this class become abstract, if you not override all the
//abstract methods of abstract class, your class become
//abstract too.
}
```


ABESIT COLLEGE OF ENGINEERING
CRC-Training

Test Your Self

1) What will be the output of the following code?

```
class Account
{
    abstract void deposit();
}
class SavingAccount extends Account
{
    void deposit()
    {
        System.out.println("Deposit Call ");
    }
    public static void main(String args[])
    {
        SavingAccount obj=new SavingAccount();
        obj.deposit();
    }
}
```

ABESIT COLLEGE OF ENGINEERING
CRC-Training

Test Your Self

2) What will be the output of the following code?

```
abstract class Account
{
void deposit()
{
      System.out.println("Abstract class Deposit Call ");
}
}

class SavingAccount extends Account
{
void deposit()
{
      System.out.println("Deposit Call ");
}
public static void main(String args[])
{
      SavingAccount obj=new SavingAccount();
      obj.deposit();
} }
```


ABESIT COLLEGE OF ENGINEERING
CRC-Training

Test Your Self

3) What will be the output of the following code?

```
abstract class Account
{
abstract void deposit();
}

class SavingAccount extends Account
{
void deposit(int a)
{
      System.out.println("Deposit Call ");
}
public static void main(String args[])
{
      SavingAccount obj=new SavingAccount();
      obj.deposit(11);

}
}
```