**Abstract Class in Java**

In Java, abstract class is declared with the abstract keyword. It may have both abstract and non-abstract methods(methods with bodies). An abstract is a Java modifier applicable for classes and methods in Java but*not for Variables*. In this article, we will learn the use of abstract classes in Java.

**What is Abstract Class in Java?**

Java abstract class is a class that can not be initiated by itself, it needs to be subclassed by another class to use its properties. An abstract class is declared using the “abstract” keyword in its class definition.

**Illustration of Abstract class**

abstract class Shape   
{  
 int color;  
 // An abstract function  
 abstract void draw();  
}

In Java, the following some *important observations*about abstract classes are as follows:

1. An instance of an abstract class can not be created.
2. Constructors are allowed.
3. We can have an abstract class without any abstract method.
4. There can be a **final method** in abstract class but any abstract method in class(abstract class) can not be declared as final  or in simpler terms final method can not be abstract itself as it will yield an error: “Illegal combination of modifiers: abstract and final”
5. We can define static methods in an abstract class
6. We can use the **abstract keyword** for declaring ***top-level classes (Outer class) as well as inner classes*** as abstract
7. If a**class** contains at least **one abstract method**then compulsory should declare a class as abstract
8. If the**Child class** is unable to provide implementation to all abstract methods of the**Parent class**then we should declare that **Child class as abstract**so that the next level Child class should provide implementation to the remaining abstract method

**Examples of Java Abstract Class**

**1. Example of Abstract Class that has Abstract method**

**Below is the implementation of the above topic:**

|  |
| --- |
| // Abstract class  abstract class Sunstar {      abstract void printInfo();  }  // Abstraction performed using extends  class Employee extends Sunstar {      void printInfo()      {          String name = "avinash";          int age = 21;          float salary = 222.2F;          System.out.println(name);          System.out.println(age);          System.out.println(salary);      }  }  // Base class  class Base {      public static void main(String args[])      {          Sunstar s = new Employee();          s.printInfo();      }  } |

**Output**

avinash

21

222.2

**Abstraction in Java**

In Java, abstraction is achieved by**interfaces**and **abstract classes**. We can achieve 100% abstraction using interfaces.

Data Abstraction may also be defined as the process of identifying only the required characteristics of an object ignoring the irrelevant details. The properties and behaviours of an object differentiate it from other objects of similar type and also help in classifying/grouping the objects.

**Abstraction Real-Life Example:**

Consider a real-life example of a man driving a car. The man only knows that pressing the accelerators will increase the speed of a car or applying brakes will stop the car, but he does not know how on pressing the accelerator the speed is actually increasing, he does not know about the inner mechanism of the car or the implementation of the accelerator, brakes, etc in the car. This is what abstraction is.

**Java Abstract classes and Java Abstract methods**

1. An abstract class is a class that is declared with an abstract keyword.
2. An abstract method is a method that is declared without implementation.
3. An abstract class may or may not have all abstract methods. Some of them can be concrete methods
4. A method-defined abstract must always be redefined in the subclass, thus making overriding compulsory or making the subclass itself abstract.
5. Any class that contains one or more abstract methods must also be declared with an abstract keyword.
6. There can be no object of an abstract class. That is, an abstract class can not be directly instantiated with the *new operator*.
7. An abstract class can have parameterized constructors and the default constructor is always present in an abstract class.

**Algorithm to implement abstraction in Java**

1. Determine the classes or interfaces that will be part of the abstraction.
2. Create an abstract class or interface that defines the common behaviours and properties of these classes.
3. Define abstract methods within the abstract class or interface that do not have any implementation details.
4. Implement concrete classes that extend the abstract class or implement the interface.
5. Override the abstract methods in the concrete classes to provide their specific implementations.
6. Use the concrete classes to implement the program logic.

**Interface**

Interfaces are another method of implementing abstraction in Java. The key difference is that, by using interfaces, we can achieve 100% abstraction in Java classes. In Java or any other language, interfaces include both methods and variables but lack a method body. Apart from abstraction, interfaces can also be used to implement interfaces in Java.

***Implementation:****To implement an****interface****we use the keyword “****implements” with class.***

|  |
| --- |
| // Define an interface named Shape  interface Shape {      double calculateArea(); // Abstract method for                              // calculating the area  }  // Implement the interface in a class named Circle  class Circle implements Shape {      private double radius;      // Constructor for Circle      public Circle(double radius) { this.radius = radius; }      // Implementing the abstract method from the Shape      // interface      public double calculateArea()      {          return Math.PI \* radius \* radius;      }  }  // Implement the interface in a class named Rectangle  class Rectangle implements Shape {      private double length;      private double width;        // Constructor for Rectangle      public Rectangle(double length, double width)      {          this.length = length;          this.width = width;      }      // Implementing the abstract method from the Shape      // interface      public double calculateArea() { return length \* width; }  }  // Main class to test the program  public class Main {      public static void main(String[] args)      {          // Creating instances of Circle and Rectangle          Circle myCircle = new Circle(5.0);          Rectangle myRectangle = new Rectangle(4.0, 6.0);            // Calculating and printing the areas          System.out.println("Area of Circle: "                             + myCircle.calculateArea());          System.out.println("Area of Rectangle: "                             + myRectangle.calculateArea());      }  } |

**Output**

Area of Circle: 78.53981633974483

Area of Rectangle: 24.0

**Advantages of Abstraction**

Here are some advantages of abstraction:

1. It reduces the complexity of viewing things.
2. Avoids code duplication and increases reusability.
3. Helps to increase the security of an application or program as only essential details are provided to the user.
4. It improves the maintainability of the application.
5. It improves the modularity of the application.
6. The enhancement will become very easy because without affecting end-users we can able to perform any type of changes in our internal system.
7. Improves code reusability and maintainability.
8. Hides implementation details and exposes only relevant information.
9. Provides a clear and simple interface to the user.
10. Increases security by preventing access to internal class details.
11. Supports modularity, as complex systems can be divided into smaller and more manageable parts.
12. Abstraction provides a way to hide the complexity of implementation details from the user, making it easier to understand and use.
13. Abstraction allows for flexibility in the implementation of a program, as changes to the underlying implementation details can be made without affecting the user-facing interface.
14. Abstraction enables modularity and separation of concerns, making code more maintainable and easier to debug.

**Disadvantages of Abstraction in Java**

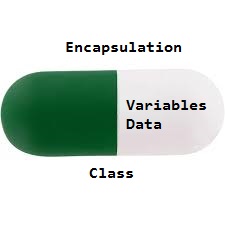
Here are the main disadvantages of abstraction in Java:

1. Abstraction can make it more difficult to understand how the system works.
2. It can lead to increased complexity, especially if not used properly.
3. This may limit the flexibility of the implementation.
4. Abstraction can add unnecessary complexity to code if not used appropriately, leading to increased development time and effort.
5. Abstraction can make it harder to debug and understand code, particularly for those unfamiliar with the abstraction layers and implementation details.
6. Overuse of abstraction can result in decreased performance due to the additional layers of code and indirection.

**Encapsulation in Java**

Encapsulation in Java is a fundamental concept in object-oriented programming (OOP) that refers to the bundling of data and methods that operate on that data within a single unit, which is called a class in Java. Java Encapsulation is a way of hiding the implementation details of a class from outside access and only exposing a public interface that can be used to interact with the class.

In Java, encapsulation is achieved by declaring the instance variables of a class as private, which means they can only be accessed within the class. To allow outside access to the instance variables, public methods called getters and setters are defined, which are used to retrieve and modify the values of the instance variables, respectively. By using getters and setters, the class can enforce its own data validation rules and ensure that its internal state remains consistent.



**Implementation of Java Encapsulation**

Below is the example with Java Encapsulation:

|  |
| --- |
| // Person Class  class Person {      // Encapsulating the name and age      // only approachable and used using      // methods defined      private String name;      private int age;      public String getName() { return name; }      public void setName(String name) { this.name = name; }      public int getAge() { return age; }      public void setAge(int age) { this.age = age; }  }  // Driver Class  public class Main {      // main function      public static void main(String[] args)      {          // person object created          Person person = new Person();          person.setName("John");          person.setAge(30);          // Using methods to get the values from the          // variables          System.out.println("Name: " + person.getName());          System.out.println("Age: " + person.getAge());      }  } |

**Output**

Name: John

Age: 30

**Encapsulation** is defined as the wrapping up of data under a single unit. It is the mechanism that binds together code and the data it manipulates. Another way to think about encapsulation is, that it is a protective shield that prevents the data from being accessed by the code outside this shield.

* Technically in encapsulation, the variables or data of a class is hidden from any other class and can be accessed only through any member function of its own class in which it is declared.
* As in encapsulation, the data in a class is hidden from other classes using the data hiding concept which is achieved by making the members or methods of a class private, and the class is exposed to the end-user or the world without providing any details behind implementation using the abstraction concept, so it is also known as a **combination of data-hiding and abstraction**.
* Encapsulation can be achieved by Declaring all the variables in the class as private and writing public methods in the class to set and get the values of variables.
* It is more defined with the setter and getter method.

**Advantages of Encapsulation**

* **Data Hiding:**it is a way of restricting the access of our data members by hiding the implementation details. Encapsulation also provides a way for data hiding. The user will have no idea about the inner implementation of the class. It will not be visible to the user how the class is storing values in the variables. The user will only know that we are passing the values to a setter method and variables are getting initialized with that value.
* **Increased Flexibility:** We can make the variables of the class read-only or write-only depending on our requirements. If we wish to make the variables read-only then we have to omit the setter methods like setName(), setAge(), etc. from the above program or if we wish to make the variables write-only then we have to omit the get methods like getName(), getAge(), etc. from the above program
* **Reusability:** Encapsulation also improves the re-usability and is easy to change with new requirements.
* **Testing code is easy:** Encapsulated code is easy to test for unit testing.
* **Freedom to programmer in implementing the details of the system:**This is one of the major advantage of encapsulation that it gives the programmer freedom in implementing the details of a system. The only constraint on the programmer is to maintain the abstract interface that outsiders see.

**Disadvantages of Encapsulation in Java**

* Can lead to increased complexity, especially if not used properly.
* Can make it more difficult to understand how the system works.
* May limit the flexibility of the implementation.

**Difference b/w encapsulation and abstration:**

| **Encapsulation** | **Abstraction** |
| --- | --- |
| Encapsulation is data hiding(information hiding) | Abstraction is detailed hiding(implementation hiding). |
| Encapsulation groups together data and methods that act upon the data | Data Abstraction deal with exposing the interface to the user and hiding the details of implementation |
| Encapsulated classes are Java classes that follow data hiding and abstraction | Implementation of abstraction is done using abstract classes and interface |
| Encapsulation is a procedure that takes place at the implementation level | abstraction is a design-level process |

**Difference b/w Abstract Classes and Interfaces in Java:**

| **Abstract Class** | **Interfaces** |
| --- | --- |
| Abstract classes support abstract and Non-abstract methods | Interface supports have abstract methods only. |
| Doesn’t support Multiple Inheritance | Supports Multiple Inheritance |
| Abstract classes can be extended by Java classes and multiple interfaces | The interface can be extended by Java interface only. |
| Abstract class members in Java can be private, protected, etc. | Interfaces are public by default. |
| **Example:**  public abstract class Vechicle{ public abstract void drive() } | **Example:**  public interface Animal{ void speak(); } |