Module 4: Abstraction

Prof. Pramod Bide

Concept of Abstraction

- Abstraction in Java is the process in which we only show essential details/functionality to the user. The non-essential implementation details are not displayed to the user. the ability of a message to be displayed in more than one form.

Real-life Illustration of Abstraction in Java: 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.

Concept of Abstraction

- 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.

Ways to achieve Abstraction

There are two ways to achieve abstraction in java

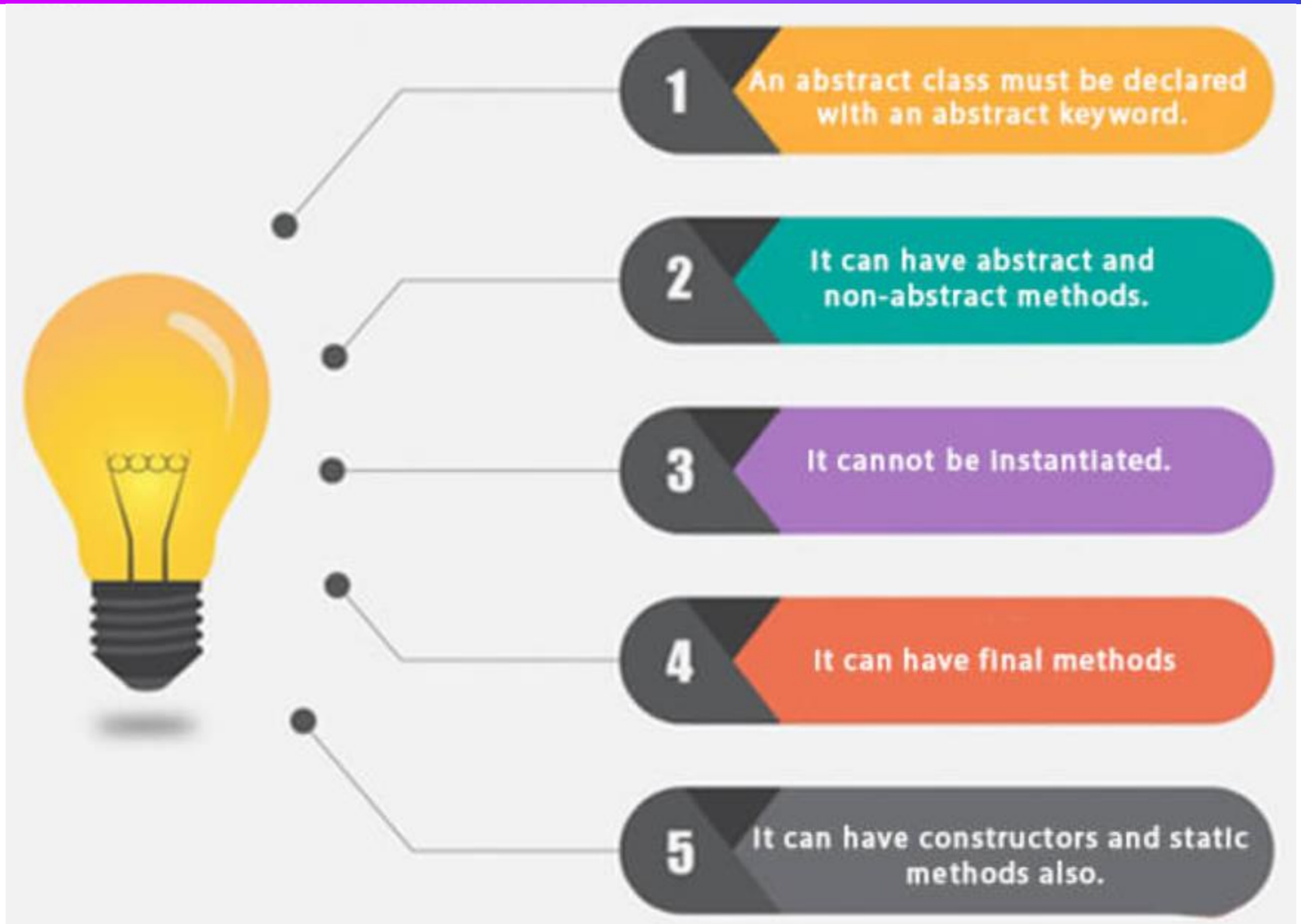
- Abstract class (0 to 100%)
- Interface (100%)

Abstract class in Java

- A class which is declared as abstract is known as an abstract class.
- It can have abstract and non-abstract methods.
- It needs to be extended and its method implemented.
- It cannot be instantiated.
- Example:

```
abstract class A{}
```

Rules for Abstract class



Abstract Method in Java

- A method which is declared as abstract and does not have implementation is known as an abstract method.
- Example:

```
abstract void printStatus();//no method body and abstract
```

Example of Abstract class

Code

```
abstract class Bike{
  abstract void run();
}
class Honda4 extends Bike{
void run(){System.out.println("running safely");}
public static void main(String args[]){
 Bike obj = new Honda4();
 obj.run();
}
}
```

Output

```
running safely
```

- In this example, Bike is an abstract class that contains only one abstract method run. Its implementation is provided by the Honda class.

Example of Abstract class

Code

```
abstract class Bank{
abstract int getRateOfInterest();
}
class SBI extends Bank{
int getRateOfInterest(){return 7;}
}
class PNB extends Bank{
int getRateOfInterest(){return 8;}
}

class TestBank{
public static void main(String args[]){
Bank b;
b=new SBI();
System.out.println("Rate of Interest is: "+b.getRateOfInterest()+" %");
b=new PNB();
System.out.println("Rate of Interest is: "+b.getRateOfInterest()+" %");
}}
```

Output

```
Rate of Interest is: 7 %
Rate of Interest is: 8 %
```

- The code defines an abstract class Bank with an abstract method getRateOfInterest(), which is implemented by concrete subclasses SBI and PNB.
- The TestBank class demonstrates polymorphism by using a reference variable of type Bank to call the getRateOfInterest() method on instances of both subclasses.

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

Example of Interface

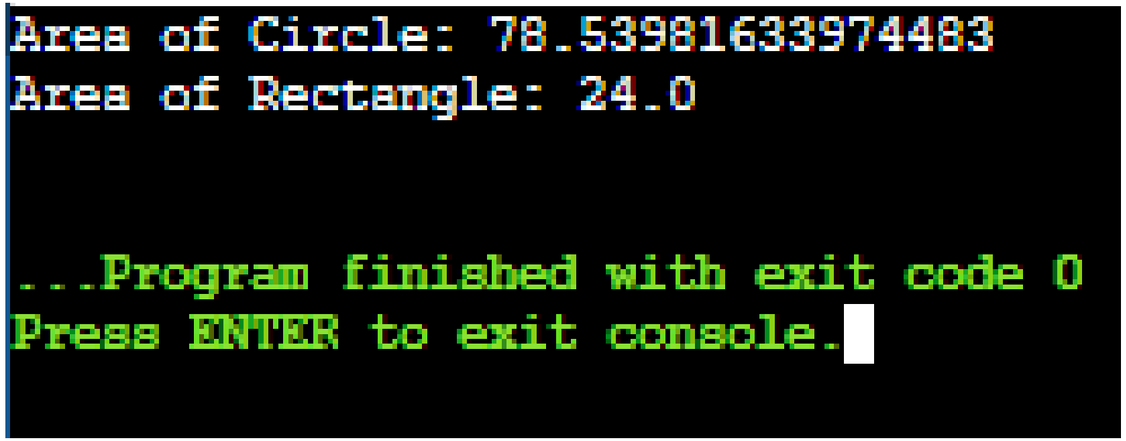
Code

```java
// 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;
    }
}
```

```java

// 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());
    }
}
```

Example of Interface

Output



- The code defines an interface named Shape with a method calculateArea(), which is implemented by the Circle and Rectangle classes
- By implementing the Shape interface, these classes promise to provide functionality for calculating the area, enabling polymorphic behavior when calculating the areas of circles and rectangles.

Advantages of Abstraction

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

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.