**1. What is an abstract class in Java? Explain with an example.**

In Java, an abstract class is a class that cannot be instantiated on its own. It is designed to be a base class (superclass) for other classes. Abstract classes may contain both abstract methods (methods without a body) and concrete methods (methods with a body).

Here's an example:

abstract class Shape { // Abstract class  
 abstract double area(); // Abstract method  
   
 public void displayColor(String color) { // Concrete method  
 System.out.println("The color of the shape is " + color);  
 }  
}

In this example, Shape is an abstract class. It has an abstract method area() and a concrete method displayColor().

**2. How is an abstract method different from a regular method in Java?**



| **Feature** | **Abstract Method** | **Regular Method** |
| --- | --- | --- |
| Definition | Declared with the abstract keyword; no body. | Declared without abstract; contains a method body. |
| Example | abstract void draw(); | void printMessage() { System.out.println("Hello"); } |
| Purpose | To define a method that *must* be implemented by subclasses. | To define a method that performs a specific action. |
| Class Context | Can only exist within an abstract class or interface. | Can exist in any class (abstract or concrete). |
| Implementation | No implementation is provided in the abstract class. | Implementation is provided in the method declaration. |

**3. Can we create an object of an abstract class? Justify your answer with an example.**

No, you cannot create an object of an abstract class directly in Java. Abstract classes are incomplete because they may contain abstract methods that lack implementation. Instantiation requires all methods to be defined.

Here's an example to illustrate:

abstract class MyAbstractClass {  
 abstract void myMethod();  
}  
  
public class Main {  
 public static void main(String[] args) {  
 // MyAbstractClass obj = new MyAbstractClass(); // This line would cause an error  
 // You cannot instantiate MyAbstractClass  
 }  
}

The commented-out line would result in a compilation error because you cannot create an instance of MyAbstractClass. You can only create instances of concrete (non-abstract) classes that inherit from the abstract class and provide implementations for all its abstract methods.

**4. Write a Java program that demonstrates the use of an abstract class and abstract method.**

abstract class Shape {  
 abstract double area();  
  
 public void displayShapeName(String shapeName) {  
 System.out.println("This is a " + shapeName);  
 }  
}  
  
class Circle extends Shape {  
 private double radius;  
  
 public Circle(double radius) {  
 this.radius = radius;  
 }  
  
 @Override  
 public double area() {  
 return Math.PI \* radius \* radius;  
 }  
}  
  
class Rectangle extends Shape {  
 private double length;  
 private double width;  
  
 public Rectangle(double length, double width) {  
 this.length = length;  
 this.width = width;  
 }  
  
 @Override  
 public double area() {  
 return length \* width;  
 }  
}  
  
public class Main {  
 public static void main(String[] args) {  
 Circle circle = new Circle(5);  
 Rectangle rectangle = new Rectangle(4, 6);  
  
 circle.displayShapeName("Circle");  
 System.out.println("Circle Area: " + circle.area());  
  
 rectangle.displayShapeName("Rectangle");  
 System.out.println("Rectangle Area: " + rectangle.area());  
 }  
}

**Explanation:**

* Shape is an abstract class with an abstract method area().
* Circle and Rectangle are concrete classes that inherit from Shape and implement the area() method.
* Objects of Circle and Rectangle can be created, and their area() methods can be called.

**5. What are the key differences between an interface and an abstract class in Java?**

| **Feature** | **Abstract Class** | **Interface** |
| --- | --- | --- |
| Keyword | abstract class | interface |
| Instantiation | Cannot be instantiated. | Cannot be instantiated. |
| Inheritance | A class can inherit from only one abstract class. | A class can implement multiple interfaces. |
| Method Declaration | Can have both abstract and concrete methods. | Prior to Java 8, could only have abstract methods. From Java 8 onwards, default and static methods are allowed. |
| Variable Declaration | Can have instance variables (both final and non-final). | Can only have constant (final, static) variables. |
| Constructor | Can have constructors. | Cannot have constructors. |
| Default Behavior | Provides a common base implementation; defines a "is-a" relationship. | Specifies a contract; defines a set of methods that a class must implement. |
| Use Case | Used when there is a clear hierarchy and some default behavior is needed. | Used to achieve complete abstraction and support multiple inheritances of behavior. |