Core Java Notes

Topic – Abstraction (Implementation Hiding)

What is Abstraction?

Abstraction is a concept in Java that allows you to hide complex implementation details and show only the essential features of an object. It helps make programs easier to understand and maintain.

How Abstraction is Achieved in Java

1. **Abstract Classes**
2. **Interfaces**

1. Abstract Classes

- **Definition**: An abstract class is a class that cannot be instantiated directly. It can have abstract methods (without implementation) and concrete methods (with implementation).
- **Purpose**: To provide a base for other classes to extend and implement specific details.
- **Example**:

```
abstract class cars
{
    abstract void MakeSound();

void show()
{
    System.out.println("Concrete method");
}
}
class FourWheeler extends cars
{
    void MakeSound()
    {
        System.out.println("Abstract method call");
    }
}
    public class AbstractClassEx
    {
        public static void main(String args[])
        {
            cars cs=new FourWheeler();
            cs.MakeSound();
            cs.show();
```

```
        }
    }
```

2. Interfaces

- **Definition**: An interface is a reference type, similar to a class, that can contain only method signatures and final variables. It cannot have instance fields or constructors.
- **Purpose**: To define a contract that other classes must follow. Interfaces support multiple inheritance (a class can implement multiple interfaces).
- **Example**:

```
interface Vacations
{
    void Holidays();
default void EnjoyHolidays()
{
    System.out.println("Hey Everyone!!");
}
}
class Holi implements Vacations
{
    public void Holidays()
    {
        System.out.println("Happy Holi,Enjoy Holiday!!");
    }
}
public class InterfaceEx
{
    public static void main(String args[])
    {
        Vacations VS=new Holi();
        VS.EnjoyHolidays();
        VS.Holidays();
    }
}
```

Key Differences Between Abstract Classes and Interfaces

Feature	Abstract Class	Interface
Methods	Can have abstract and concrete methods	Methods are abstract by default; can have default methods
Fields	Can have instance variables	Can only have public static final variables (constants)
Constructors	Can have constructors	Cannot have constructors
Inheritance	Single inheritance (one abstract class)	Multiple inheritance (multiple interfaces)

Feature	Abstract Class	Interface
Access Modifiers	Can have various access modifiers	Methods are implicitly public; fields are public static final

When to Use Each

- **Abstract Classes**: Use when you want to provide a common base with shared code and some default behavior. Subclasses will extend this class and add their own specific implementation.
- **Interfaces**: Use when you want to define a set of methods that can be implemented by any class. Interfaces are useful for defining capabilities or behaviors that can be shared across unrelated classes.