Ans1

**Key Components of JDK (Java Development Kit)**

The **JDK** is a complete software development kit needed for developing Java applications. It includes everything you need to write, compile, debug, and run Java programs. Its key components are:

**1. Java Compiler (javac)**

* **Purpose:** Converts Java source code (.java files) into bytecode (.class files).
* **Explanation:** Bytecode is a special machine-independent code that can run on any platform that has a Java Virtual Machine (JVM).
* **Example:** Running javac HelloWorld.java compiles your code.

**2. Java Runtime Environment (JRE)**

* **Purpose:** Provides the libraries, JVM, and other resources to run Java applications.
* **Explanation:** The JDK includes a **full** JRE so you can not just develop but also test your applications.

**3. Java Virtual Machine (JVM)**

* **Purpose:** Executes Java bytecode.
* **Explanation:** JVM interprets or compiles (just-in-time compilation) the bytecode into machine-specific code so that your Java program runs on any device or operating system.

**4. Java API Libraries**

* **Purpose:** Ready-made classes and functions to simplify programming.
* **Explanation:** JDK comes with many built-in libraries for data structures (like ArrayList), networking, file handling, GUI building (like Swing, JavaFX), and more.

**5. Development Tools**

* **Purpose:** Assist in developing, debugging, monitoring, and documenting Java programs.
* **Important tools:**
  + javap – Java class file disassembler (shows what's inside compiled .class files).
  + javadoc – Documentation generator from comments in code.
  + jdb – Java debugger.
  + jar – Tool for bundling Java files into .jar (Java Archive) files.
  + jconsole – Monitor Java application performance.

**6. Java Doc Tool (javadoc)**

* **Purpose:** Automatically generates HTML documentation from your Java source code.
* **Explanation:** When you comment your code with special tags (/\*\* \*/), javadoc reads them and builds organized documentation.

**7. Java Package Management (jar, jlink)**

* **Purpose:** Package and manage Java applications.
* **Explanation:**
  + jar bundles files into compressed archives.
  + jlink creates a custom runtime image containing only the modules your application needs.

Ans2 **Encapsulation**

**➔ Definition:**

**Encapsulation** is the process of wrapping data (variables) and code (methods) together into a single unit, usually a **class**, and restricting direct access to some of the object's components.

**➔ Key Points:**

* Data hiding: Private variables cannot be accessed directly from outside the class.
* Access through **getters** and **setters** (public methods).
* Helps in protecting the internal state of an object.

**➔ Example:**

class Person {

private String name; // private data

// Getter method

public String getName() {

return name;

}

// Setter method

public void setName(String newName) {

name = newName;

}

}

Here, name is hidden from direct access. Outside code must use getName() or setName().

**Polymorphism**

**➔ Definition:**

**Polymorphism** means "many forms". In Java, it allows objects to behave differently based on their actual class, even if they share the same interface or superclass.

**➔ Key Points:**

* **Compile-time Polymorphism** (Method Overloading): Same method name with different parameters.
* **Runtime Polymorphism** (Method Overriding): A subclass provides a specific implementation of a method that is already defined in its superclass.
* Makes code flexible and easier to extend.

**➔ Example of Method Overriding:**

class Animal {

void sound() {

System.out.println("Animal makes a sound");

}

}

class Dog extends Animal {

@Override

void sound() {

System.out.println("Dog barks");

}

}

Animal obj = new Dog();

obj.sound(); // Output: Dog barks

Even though obj is of type Animal, it calls the sound() method of Dog at runtime — that's polymorphism!

Ans3 **Inheritance in Java**

**➔ Definition:**

**Inheritance** is a mechanism in Java where one class (**child** or **subclass**) inherits the properties (fields) and behaviors (methods) of another class (**parent** or **superclass**).

It promotes **code reusability** — you don’t have to write the same code again!

**Key Concepts:**

* The extends keyword is used to inherit a class.
* A subclass can **use**, **override**, or **extend** the behavior of the superclass.
* **Single inheritance** is supported in Java (one class can inherit only one class directly).
* **Multilevel inheritance** is allowed (A → B → C).
* **Multiple inheritance** (from multiple classes) is not allowed through classes (but possible through interfaces).

**Simple Example**

// Superclass

class Animal {

void eat() {

System.out.println("This animal eats food.");

}

}

// Subclass

class Dog extends Animal {

void bark() {

System.out.println("The dog barks.");

}

}

// Main class to test

public class TestInheritance {

public static void main(String[] args) {

Dog myDog = new Dog();

myDog.eat(); // Inherited method

myDog.bark(); // Own method

}

}