Ans1 The Java Development Kit (JDK) is a full package needed for Java programming. It includes:

- **Java Compiler (javac):** Changes Java code into bytecode that the computer can understand.
- **Java Virtual Machine (JVM):** Runs the bytecode and lets Java programs work on any system.
- **Java Runtime Environment (JRE):** Has the JVM and important libraries needed to run Java programs.
- **Development Tools:** Tools like javac, java, javadoc, jdb, etc., that help with writing, compiling, and fixing Java programs.

Extra components:

- **Java API:** Ready-to-use libraries and classes to make coding easier.
- **JavaFX and Swing:** Libraries used to design graphical user interfaces (GUIs).
- **Java Native Interface (JNI):** Helps Java programs connect with code written in other languages.
- **Java Mission Control and Flight Recorder:** Used for checking and improving the performance of Java programs.
- **JAR files:** Used to bundle Java code and files into one easy-to-share package.

Ans2 Encapsulation is an important idea in Object-Oriented Programming (OOP) where data and related methods are grouped together inside a class. This makes the program easier to manage and protects the data from being changed accidentally. Encapsulation also improves modularity and reuse of code.

Polymorphism comes from Greek words meaning "many forms." In OOP, it means that the same function or method can work in different ways depending on the object it is working on.

There are two types of polymorphism:

- **Compile-time Polymorphism (Method Overloading):** Happens when there are several methods with the same name but different parameters in one class. The compiler figures out which one to call.

- **Runtime Polymorphism (Method Overriding):** Happens when a subclass changes the method of its parent class. Which method to run is decided while the program is running.

Ans3 Inheritance is when one class (called a child or subclass) gets properties and behaviors from another class (called the parent or superclass). It helps to avoid writing the same code again and supports a hierarchy between classes.

Example:

java

```
// Parent class: Person
class Person {
      String name;
      int age;
```

```
	Person(String name, int age) {
		this.name = name;
		this.age = age;
	}

	void showInfo() {
		System.out.println("Name: " + name);
		System.out.println("Age: " + age);
	}
}

// Child class: Student inherits from Person
class Student extends Person {
	int studentId;

	Student(String name, int age, int studentId) {
		super(name, age); // Calling parent constructor
		this.studentId = studentId;
	}

	// Overriding showInfo method
	@Override
	void showInfo() {
		super.showInfo(); // Calling parent method
		System.out.println("Student ID: " + studentId);
	}
}

// Main class to test
public class TestInheritance {
	public static void main(String[] args) {
		Student student1 = new Student("Anjana", 20, 12);
		student1.showInfo();
	}
}
```