Ans1 The Java Development Kit (JDK) comprises essential components for Java development:

- Java Compiler (javac): Translates Java source code to bytecode.
- Java Virtual Machine (JVM): Executes Java bytecode, ensuring platform independence.
- Java Runtime Environment (JRE): Includes JVM and libraries for running Java applications.
- Development Tools: javac, java, javap, javadoc, jdb, etc., for coding, compiling, and debugging.

And other (extra)

- Java API: Pre-built classes and libraries for common functionalities.
- JavaFX and Java Swing: GUI libraries for building graphical interfaces.
- Java Native Interface (JNI): Allows Java code to interact with other languages.
- Java Mission Control and Flight Recorder: Tools for monitoring and profiling Java applications.
- JAR files: Packaged format for distributing Java classes and resources.

Ans2 Encapsulation is a concept in Object-Oriented Programming (OOP) that involves bundling data and methods that operate on that data into a single unit, known as a class. This unit acts as a blueprint for creating objects. Encapsulation provides benefits such as data hiding, modularity, and code reusability. The term "polymorphism" is derived from the Greek words "poly," meaning many, and "morphē," meaning forms. In the context of programming and Object-Oriented Programming (OOP), polymorphism refers to the concept that a single interface or method can be used to represent and work with objects of various types.There are two main types of polymorphism:Compile-time Polymorphism (Method Overloading):Involves having multiple methods with the same name in a class, each with a different parameter list (types or number of parameters). The appropriate method is determined by the compiler during compilation.Runtime Polymorphism (Method Overriding):Occurs when a subclass provides a specific implementation for a method that is already defined in its superclass. The decision on which method to invoke is made at runtime based on the actual type of the object.

Ans3 Inheritance allows a class (subclass or derived class) to inherit properties and behaviors from another class (superclass or base class). The subclass can reuse, extend, or override the functionalities of the superclass, promoting code reuse and creating a hierarchical relationship between classes.

Example:

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

        public Person(String name, int age) {
        this.name = name;
        this.age = age;
        }
    void displayDetails() {
        System.out.println("Name: " + name);
        System.out.println("Age: " + age);
        }
}

// Subclass: Student (inherits from Person)
class Student extends Person {
        int studentId;

        public Student(String name, int age, int studentId) {
        super(name, age);  // Call the superclass constructor
        this.studentId = studentId;
        }
```

```
	// Override the displayDetails method to include additional information
	@Override
	void displayDetails() {
	super.displayDetails();  // Call the superclass method
	System.out.println("Student ID: " + studentId);
	}
}

// Main class to demonstrate inheritance
public class InheritanceExample {
	public static void main(String[] args) {
	// Creating an instance of the subclass (Student)
	Student myStudent = new Student("Anjana", 20, 12);

	// Calling the overridden method from the superclass (Person)
	myStudent.displayDetails();
	}
}
```