Java Concepts and Examples

Class and Object

A class is a blueprint or template that defines the structure and behavior of objects.

An object is an instance of a class that represents a real-world entity.

```
class Car {
    String brand;
    int year;
}

Car car1 = new Car();
car1.brand = "Toyota";
car1.year = 2020;

Car car2 = new Car();
car2.brand = "Honda";
car2.year = 2022;
```

Method

A method is a block of code that performs a specific task and can be called on an object.

```
class Calculator {
    int add(int a, int b) {
        return a + b;
    }
}

Calculator calc = new Calculator();
int result = calc.add(5, 3); // result is now 8;
```

Abstraction

Abstraction is the concept of hiding complex implementation details and showing only the essential features of an object.

```
abstract class Shape {
    abstract void draw();
}

class Circle extends Shape {
    void draw() {
        System.out.println("Drawing a circle");
    }
}
```

Method Overloading and Overriding

Method Overloading: Defining multiple methods in the same class with the same name but different parameter lists.
Method Overriding: Overriding a method in a subclass to provide a specific implementation.

```
class Animal {
    void makeSound() {
        System.out.println("Animal makes a sound");
    }
}

class Dog extends Animal {
    void makeSound() {
        System.out.println("Dog barks");
    }
}
```

Constructor Overloading and Overriding

Constructor Overloading: Defining multiple constructors in a class with different parameter lists.
Constructor Overriding: Constructors are not overridden like methods; they are called in the order of class hierarchy.

```
class Person {
    String name;

    Person(String n) {
        name = n;
    }
}

class Student extends Person {
    int rollNo;

    Student(String n, int r) {
        super(n);
        rollNo = r;
    }
}
```

Interface

An interface defines a contract that classes must adhere to. It contains method signatures without implementations.

```
interface Printable {
    void print();
}

class Document implements Printable {
    void print() {
        System.out.println("Printing document");
    }
}
```

Inheritance

Inheritance allows a class to inherit properties and behaviors from another class (parent or superclass).

```
class Vehicle {
    void start() {
        System.out.println("Vehicle starting");
    }
}

class Car extends Vehicle {
    // Car inherits the 'start' method
}
```

Multi-Threading

Multi-threading enables a program to execute multiple threads concurrently, improving performance.

```
class MyThread extends Thread {
    public void run() {
        System.out.println("Thread is running");
    }
}

public class Main {
    public static void main(String[] args) {
        MyThread thread = new MyThread();
        thread.start(); // Starts the new thread
    }
}
```

Exception Handling

Exception handling allows you to handle errors and unexpected situations gracefully.

```
try {
    int result = 10 / 0; // This will throw an ArithmeticException
} catch (ArithmeticException e) {
    System.out.println("Error: " + e.getMessage());
}
```

Collection

Collections are Java classes that provide convenient ways to work with groups of objects.

```
import java.util.ArrayList;

public class Main {
    public static void main(String[] args) {
        ArrayList<String> names = new ArrayList<>();
        names.add("Alice");
        names.add("Bob");
        names.add("Charlie");

        for (String name : names) {
            System.out.println(name);
        }
    }
}
```