**Unit-5 Notes**

**Classes and Objects in Java**

**Concepts of Classes and Objects**

* **Class**: A blueprint or prototype from which objects are created. It can contain fields (variables) and methods to define the behavior of the objects.
* **Object**: An instance of a class. When a class is instantiated, memory is allocated for the instance variables of the class.

**Example:**

class Car {

// Fields

String color;

String model;

int year;

// Methods

void displayDetails() {

System.out.println("Model: " + model);

System.out.println("Color: " + color);

System.out.println("Year: " + year);

}

}

public class Main {

public static void main(String[] args) {

// Creating an object

Car myCar = new Car();

myCar.color = "Red";

myCar.model = "Toyota";

myCar.year = 2020;

myCar.displayDetails();

}

}

**Declaring Objects**

To declare an object, you use the class name followed by the object name.

**Example:**

Car myCar; // Declares an object of type Car

**Assigning Object Reference Variables**

Assigning a new instance of a class to an object reference variable.

**Example:**

Car myCar = new Car();

**Methods**

A method is a block of code that performs a specific task. Methods are defined within a class and called on objects.

**Example:**

class Car {

// Method

void startEngine() {

System.out.println("Engine started.");

}

}

**Constructors**

A constructor is a special method that is called when an object is instantiated. It initializes the object.

**Example:**

class Car {

String color;

String model;

// Constructor

Car(String color, String model) {

this.color = color;

this.model = model;

}

}

**Access Control**

Java provides access control through access modifiers: public, protected, private, and default (no modifier).

* **Public**: Accessible from any other class.
* **Private**: Accessible only within the declared class.
* **Protected**: Accessible within the same package and subclasses.
* **Default**: Accessible only within the same package.

**Example:**

class Car {

private String color;

public String model;

private void setColor(String color) {

this.color = color;

}

}

**Garbage Collection**

Java has an automatic garbage collection process to manage memory by reclaiming memory occupied by objects that are no longer in use.

**Example:**

class Car {

@Override

protected void finalize() {

System.out.println("Car object is being garbage collected.");

}

}

**Usage of static with Data and Methods**

* **Static Data**: Shared among all instances of a class.
* **Static Methods**: Belong to the class rather than any specific instance and can be called without creating an object.

**Example:**

class Car {

static int numberOfCars;

static void incrementCars() {

numberOfCars++;

}

}

**Usage of final with Data**

* **Final Variables**: Values cannot be changed once assigned.
* **Final Methods**: Cannot be overridden.
* **Final Classes**: Cannot be subclassed.

**Example:**

class Car {

final String VIN;

Car(String vin) {

this.VIN = vin;

}

}

**Overloading Methods and Constructors**

* **Method Overloading**: Same method name with different parameters.
* **Constructor Overloading**: Same constructor name with different parameters.

**Example:**

class Car {

void start() {

System.out.println("Car started.");

}

void start(String keyType) {

System.out.println("Car started with " + keyType);

}

Car() {

// Default constructor

}

Car(String color, String model) {

// Parameterized constructor

}

}

**Parameter Passing - Call by Value**

In Java, primitive data types are passed by value. Objects are passed by reference value, meaning the reference itself is passed by value.

**Example:**

class Car {

String color;

void changeColor(Car car, String newColor) {

car.color = newColor;

}

}

public class Main {

public static void main(String[] args) {

Car myCar = new Car();

myCar.color = "Red";

Car anotherCar = new Car();

anotherCar.changeColor(myCar, "Blue");

System.out.println(myCar.color); // Outputs "Blue"

}

}

**Recursion**

A method that calls itself.

**Example:**

class Factorial {

int factorial(int n) {

if (n == 0) {

return 1;

} else {

return n \* factorial(n - 1);

}

}

}

**Nested Classes**

Classes defined within another class.

* **Member Inner Class**: Non-static nested class.
* **Static Nested Class**: Static nested class.
* **Local Inner Class**: Defined within a block.
* **Anonymous Inner Class**: Class without a name.

**Example:**

class Outer {

class Inner {

void display() {

System.out.println("Inner class method.");

}

}

static class StaticInner {

void display() {

System.out.println("Static inner class method.");

}

}

}

**Complete Example:**

class Car {

// Fields

private String color;

private String model;

private int year;

static int numberOfCars;

// Default constructor

public Car() {

numberOfCars++;

}

// Parameterized constructor

public Car(String color, String model, int year) {

this.color = color;

this.model = model;

this.year = year;

numberOfCars++;

}

// Getter and Setter

public String getColor() {

return color;

}

public void setColor(String color) {

this.color = color;

}

public String getModel() {

return model;

}

public void setModel(String model) {

this.model = model;

}

public int getYear() {

return year;

}

public void setYear(int year) {

this.year = year;

}

// Static method

public static int getNumberOfCars() {

return numberOfCars;

}

// Method overloading

public void start() {

System.out.println("Car started.");

}

public void start(String keyType) {

System.out.println("Car started with " + keyType);

}

// Inner class

class Engine {

void display() {

System.out.println("Engine is running.");

}

}

// Static nested class

static class Maintenance {

void perform() {

System.out.println("Maintenance performed.");

}

}

// Recursive method example

public int factorial(int n) {

if (n == 0) {

return 1;

} else {

return n \* factorial(n - 1);

}

}

@Override

protected void finalize() {

System.out.println("Car object is being garbage collected.");

}

public static void main(String[] args) {

Car car1 = new Car("Red", "Toyota", 2020);

Car car2 = new Car();

car1.start();

car1.start("Smart Key");

Car.Engine engine = car1.new Engine();

engine.display();

Car.Maintenance maintenance = new Car.Maintenance();

maintenance.perform();

System.out.println("Factorial of 5: " + car1.factorial(5));

System.out.println("Total number of cars: " + Car.getNumberOfCars());

}

}