Index

43. Constructor
44. Default vs Parameterized Constructor
45. Naming Convention
46. Anonymous Object
47. What is Inheritance
48. Need of Inheritance
49. Single and Multilevel inheritance
50. Multiple Inheritance
51. This and super method
52. Method Overriding
53. Packages
54. Access Modifiers
55. Polymorphism
56. Dynamic Method Dispatch
57. Final keyword
58. Object Class equals toString hashcode
59. Upcasting and Downcasting
60. Wrapper Class

Java Cheat Sheet

1. Introduction to Java

- Java is an object-oriented, platform-independent programming language used for building applications. It supports features like multi-threading, automatic memory management, and secure execution of code.

2. JDK Setup

- JDK (Java Development Kit) includes tools like the compiler (`javac`) and runtime environment (JRE).
- Installation:

1. Download the JDK from [Oracle] (https://www.oracle.com/java/technologies/javase-downloads.html).
2. Set `JAVA_HOME` environment variable to the JDK path.
3. Update `PATH` variable to include `JAVA_HOME/bin`.

3. First Code in Java

```
public class HelloWorld {
   public static void main(String[] args) {
      System.out.println("Hello,  World!");
   }}
```

- Compile: `javac HelloWorld.java`
- Run: `java HelloWorld`

4. How Java Works

- Source code is compiled to bytecode using the `javac` compiler.
- The JVM (Java Virtual Machine) interprets bytecode and executes it on any platform, making Java platform-independent.

5. Variables

- Variables store data for processing. Java supports local, instance, and static variables.
- Syntax: `dataType variableName = value;
- int age = 25;

6. Data Types

- Primitive Types**: `byte`, `short`, `int`, `long`, `float`, `double`, `char`, `boolean`
- Non-primitive Types**: Arrays, Classes, Interfaces, etc.
- **Example:**
  int num = 10; char letter = 'A';

7. Literals

- Fixed values assigned to variables.
- **Example:**
- `int a = 10;`, `double d = 10.5;`, `char c = 'A';`, `boolean b = true;`

8. Type Conversion

- Implicit Conversion: Automatic type conversion by the compiler.
- **Example:**

```
int a = 10;
double b = a; // Implicit conversion from int to double
```

- Explicit Conversion (Casting)**: Manually converting one data type to another.
- **Example:**

```
double x = 10.5;
int y = (int) x; // Casting double to int
```

9. Arithmetic Operators

- `+`, `-`, `*`, `/`, `%` (Addition, Subtraction, Multiplication, Division, Modulo)
- **Example:**

```
int sum = 10 + 5;  // sum = 15
```

10. Relational Operators

- `==`, `!=`, `>`, `<`, `>=`, `<=`
- **Example:**

```
if (5 > 3) {
    System.out.println("5 is greater than 3");
}
```

11. Logical Operators

- **Example:**
- `&&` (AND), `||` (OR), `!` (NOT)
- boolean result = (5 > 3) && (8 > 6); // result = true

12. If Else

- Executes a block of code based on a condition.
- **Example:**

```
if (age >= 18) {
    System.out.println("Eligible to vote");
} else {
    System.out.println("Not eligible to vote");
}
```

13. If Else If

- Multiple conditions are checked sequentially.
- **Example:**

```
if (marks >= 90) {
    System.out.println("A Grade");
} else if (marks >= 75) {
    System.out.println("B Grade");
} else {
    System.out.println("C  Grade");
}
```

14. Ternary Operator

- Shorthand for `if-else`.
- Syntax: `condition ? expr1 : expr2;`
- **Example:**

```
int a = 10, b = 20;
int max = (a > b) ? a : b;  // max = 20
```

15. Switch Statement

- Simplifies code that has multiple conditions.
- **Example:**

```
int day = 3;
switch (day) {
    case 1: System.out.println("Monday"); break;
    case 2: System.out.println("Tuesday"); break;
    case 3: System.out.println("Wednesday"); break;
    default: System.out.println("Invalid Day");
}
```

16. Need for Loop

- Loops help in executing a block of code repeatedly as long as the condition is true.

17. While Loop

- Repeats as long as the condition is true.
- **Example:**

```
int i = 0;
while (i < 5) {
System.out.println("i = " + i);
i++;
}
```

18. Do While Loop

- Executes at least once, then checks the condition
- **Example:**

```
int i = 0;
do {
System.out.println("i = " + i);
 i++;
} while (i < 5);
```

19. For Loop

- Used for a definite number of iterations.
- **Example:**

```
for (int i = 0; i < 5; i++) {
System.out.println("i = " + i);
}
```

20. Which Loop to Use

- For Loop: Use when the number of iterations is known.
- While Loop: Use when you want to repeat until a condition changes.
- Do-While Loop: Use when you want the loop to execute at least once.

21. Class and Object Theory

- Class: Blueprint or template to create objects.
- Object: Instance of a class with a state and behavior.
- **Example:**

```
class Car { // Class
String color; // State
void drive() { // Behavior
System.out.println("Driving...");
}}

public class Main {
public static void main(String[] args) {
Car car = new Car(); // Object creation
car.color = "Red";
car.drive();
}}
```

22. Class and Object Practical

- Create classes and objects, set fields, and call methods practically.
- **Example:**

```
class Student {
int id;
String name;
void display() {
System.out.println(id + " " + name);
} }

public class Test {
public static void main(String[] args) {
Student s1 = new Student();
s1.id = 101;
s1.name = "John";
s1.display();
} }
```

23. JDK, JRE, JVM

- JDK (Java Development Kit): Contains tools to develop Java programs.
- JRE (Java Runtime Environment): Provides libraries and JVM for running Java programs.
- JVM (Java Virtual Machine): Converts bytecode into machine code and executes it.

24. Methods

- Block of code to perform a specific task, invoked by calling it.
- **Example:**

```
class MathOperations {
int add(int a, int b) {
return a + b;
}}
public class Test {
public static void main(String[] args) {
MathOperations math = new MathOperations();
int result = math.add(10, 20);
System.out.println("Sum: " + result);
}}
```

25. Method Overloading

- Multiple methods with the same name but different parameters in the same class.
- **Example:**

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

26. Stack and Heap

- Stack: Stores local variables and function calls.
- Heap: Stores objects dynamically created during runtime.

27. Need of Array

- Arrays store multiple elements of the same data type, reducing code complexity.

28. Creation of Array

- Syntax: `dataType[] arrayName = new dataType[size];`
- **Example:**

```
int[] numbers = new int[5];
numbers[0] = 10;
```

29. Multi-Dimensional Array

- Arrays containing arrays. E.g., 2D arrays for matrix representation.
- **Example:**
  int[][] matrix = {{1, 2}, {3, 4}, {5, 6}};

30. Jagged and 3D Array

- Jagged Array: Arrays with varying row sizes.
- **Example:**
  int[][] jaggedArray = new int[3][];
  jaggedArray[0] = new int[2]; // **First row has 2 columns**
  jaggedArray[1] = new int[3]; // **Second row has 3 columns**

- 3D Array: Array with three dimensions.
- **Example:**
  int[][][] threeDArray = new int[2][3][4];

31. Drawbacks of Array

- Fixed size, unable to change dynamically.
- Only homogeneous data type storage.

32. Array of Objects

- Arrays storing object references.
- **Example:**

```
class Student {
String name;
Student(String name) { this.name = name; }
}

public class Test {
 public static void main(String[] args) {
Student[] students = new Student[2];
students[0] = new Student("Alice");
students[1] = new Student("Bob");
  }
}
```

33. Enhanced For Loop

- Iterates through arrays or collections without using an index.
- **Example:**
  int[] numbers = {10, 20, 30};
  for (int num : numbers) {
  System.out.println(num);}

34. What is String

- Sequence of characters represented as objects of the `String` class.
  String str = "Hello, World!";

35. Mutable vs Immutable String

- -Immutable: `String` - Once created, cannot be changed.
- Mutable: `StringBuilder` or `StringBuffer`.
- **Example:**
  String str = "Hello"; // **Immutable**
  StringBuilder sb = new StringBuilder("Hello"); // **Mutable**
  sb.append(" World");

36. StringBuffer and StringBuilder

- StringBuffer: Thread-safe, slower.
- StringBuilder: Non-thread-safe, faster.
- **Example:**
  StringBuilder sb = new StringBuilder("Hello");
  sb.append(" World");

37. Static Variable

- Belongs to the class rather than any object instance.
- **Example:**
  class Test {
  static int count = 0;
  }

38. Static Method

- Can access only static data members and call other static methods.
- **Example:**

```
class Test {
 static void display() {
System.out.println("Static Method");
}}
```

39. Static Block

- Executes before the main method, used for initialization.
- **Example:**

```
 class Test {
static {
System.out.println("Static  Block");
}}
```

40. Encapsulation

- Bundling data (variables) and methods in a single unit (class) and restricting access using private access modifiers.
- **Example:**

```
class Person {
private String name;
public String getName() { return name; }
public void setName(String name) { this.name = name; }
}
```

41. Getters and Setters

- Definition: Used to access and update the private fields of a class.
- **Example:**

```
public class Student {
private String name;

 // Getter
 public String getName() {
  return name;
 }

// Setter
public void setName(String name) {
this.name = name;
}}
```

42. `this` Keyword

- Definition: Refers to the current object instance.
- **Example:**

```
public class Employee {
private int id;
public Employee(int id) {
this.id = id;                // `this` distinguishes between class attribute and parameter
} }
```

43. Constructor

- Definition: A special method invoked when an object is created. Used to initialize objects.
- **Example:**

```
public class Car {
private String model;

// Constructor
public Car(String model) {
   this.model = model;
}}
```

44. Default vs Parameterized Constructor

- Default Constructor: Constructor with no arguments, provided by Java if no other constructor is defined.
- Parameterized Constructor: Takes arguments to initialize an object.
- Example:

```
public class Vehicle {
private String type;

// Default Constructor
   public Vehicle() {
   this.type = "Unknown";
   }

// Parameterized Constructor
   public Vehicle(String type) {
   this.type = type;
   }
    }
```

45. Naming Conventions

- Class Names: Use Pascal Case (e.g., `Student`, `EmployeeDetails`).
- Variable and Method Names: Use camelCase (e.g., `studentName`, `calculateTotal`).

46. Anonymous Object

- Definition: An object created without being referenced by a variable.
- **Example:**
- new Car("Toyota").displayModel();

47. What is Inheritance

- Definition: Mechanism to acquire properties and behaviors of a parent class.
- **Example:**

```
public class Animal {
void eat() { System.out.println("Eating..."); }
}

public class Dog extends Animal {
void bark() { System.out.println("Barking..."); }
}
```

48. Need of Inheritance

- Purpose: Code reusability, method overriding, and to establish a parent-child relationship.

49. Single and Multilevel Inheritance

- Single Inheritance: A class extends one superclass.
- Multilevel Inheritance: A class is derived from another derived class.
- **Example:**

```
// Single Inheritance
class A { }
class B extends A { }

// Multilevel Inheritance
class C extends B { }
```

50. Multiple Inheritance

- Definition: A class cannot extend more than one class in Java due to ambiguity (solved by interfaces).
- **Example:**

```
interface A { }
interface B { }
class C implements A, B { }
```

51. `this` and `super` Keyword

- this`: Refers to the current instance.
- `super`: Refers to the superclass instance and can invoke the superclass's constructor or methods.
- **Example:**

```
class Animal {
void sound() { System.out.println("Animal Sound"); }
}

class Dog extends Animal {
void sound() {
super.sound();          // Calls Animal's sound method
System.out.println("Dog Barks");
  }
}
```

52. Method Overriding

- Definition: Subclass has a method with the same signature as a method in its superclass.
- **Example:**

```
class Parent {
void show() { System.out.println("Parent show"); }
}

class Child extends Parent {
void show() { System.out.println("Child show"); }
}
```

53. Packages

- Definition: Grouping related classes and interfaces together.
- **Example:**

```
package com.mycompany.project;
public class MyClass { }
```

54. Access Modifiers

- Types: `public`, `private`, `protected`, and default.
- Purpose: Control visibility of classes, methods, and variables.

55. Polymorphism

- Definition: Ability to present the same interface for different data types.
- Types: Compile-time (method overloading) and Run-time (method overriding).
- **Example:**

```
class Animal {
void sound() { System.out.println("Generic Animal Sound"); }
}

class Cat extends Animal {
void sound() { System.out.println("Cat Meows"); }
}
```

56. Dynamic Method Dispatch

- Definition: Method call resolved at runtime based on the object's actual type.
- **Example:**

```
Animal a = new Cat();          // Upcasting
a.sound();                     // Calls Cat's sound method
```

57. `final` Keyword

- Definition: Used to declare constants, prevent method overriding, and inheritance.
- **Example:**

```
final class FinalClass { }              // Cannot be subclassed

class Test {
final void display() { }                // Cannot be overridden
 }
```

58. `Object` Class Methods

- `equals()`: Compares two objects for equality.
- `toString()`: Returns string representation of an object.
- `hashCode()`: Returns hash code value for the object.

59. Upcasting and Downcasting

- Upcasting: Casting a subclass type to a superclass type.
- Downcasting: Casting a superclass type to a subclass type.
- Example:
  Animal a = new Cat(); // **Upcasting**
  Cat c = (Cat) a; // **Downcasting**

60. Wrapper Class

- Definition: Provides a way to use primitive data types as objects (e.g., `Integer`, `Character`).
- **Example:**

int num = 10;
Integer obj = Integer.valueOf(num); // **Boxing**
int n = obj.intValue(); // **Unboxing**

This cheat sheet covers the basic concepts and examples to help you quickly grasp the main OOP principles in Java. Let me know if you need more details or specific examples on any of these topics!