ChatGPT

Java OOP + Basic Coding Cheat Sheet

---

1. Basic Java Structure

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

---

2. Classes & Objects

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

    // Constructor
    public Car(String brand, int year) {
        this.brand = brand;
        this.year = year;
    }

    void display() {
        System.out.println(brand + " " + year);
    }
}
```

---

3. Inheritance (extends)

```
class Animal {
    void eat() {
        System.out.println("This animal eats food.");
    }
}

class Dog extends Animal {
    void bark() {
```

```
        System.out.println("The dog barks.");
    }
}
```

---

4. Abstraction (abstract class)

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

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

---

5. Interface (implements)

```
interface Animal {
    void sound();
}

class Dog implements Animal {
    public void sound() {
        System.out.println("Bark");
    }
}
```

---

6. Arrays in Java

```
// Declaration
int[] arr = new int[5];

// Initialization
arr[0] = 10;
arr[1] = 20;

// Declaration + Initialization
int[] nums = {1, 2, 3, 4, 5};
```

```
// Looping
for(int i = 0; i < nums.length; i++) {
    System.out.println(nums[i]);
}
```

---

7. Strings in Java

```
String str = "Hello";
System.out.println(str.length());
System.out.println(str.charAt(0));
System.out.println(str.substring(1,3));
System.out.println(str.equals("Hello"));
```

---

8. Array of Strings

```
// Declaration and Initialization
String[] fruits = {"Apple", "Mango", "Banana", "Grapes"};

// Accessing Elements
System.out.println(fruits[0]); // Apple

// Looping through array
for(String fruit : fruits) {
    System.out.println(fruit);
}

// Sorting without using Arrays.sort() - Bubble Sort
for(int i = 0; i < fruits.length - 1; i++) {
    for(int j = 0; j < fruits.length - i - 1; j++) {
        if(fruits[j].compareTo(fruits[j + 1]) > 0) { // Lexicographical
comparison
            // Swap
            String temp = fruits[j];
            fruits[j] = fruits[j + 1];
            fruits[j + 1] = temp;
        }
    }
}

System.out.println("After Sorting (Bubble Sort):");
for(String fruit : fruits) {
```

```
    System.out.println(fruit);
}
```

Important Notes:

- `compareTo()` method compares two strings lexicographically (based on ASCII values).
- Returns:
- `0` if equal
- Positive if 1st string > 2nd string
- Negative if 1st string < 2nd string
- No predefined sorting functions like `Arrays.sort()` used here.

---

9. Using Arrays/Strings in Classes

```
class Student {
    String name;
    int[] marks;

    public Student(String name, int[] marks) {
        this.name = name;
        this.marks = marks;
    }

    void display() {
        System.out.print(name + " Marks: ");
        for(int m : marks) {
            System.out.print(m + " ");
        }
    }
}
```

---

10. Sorting an Array without Predefined Functions (Bubble Sort)

```
int[] numbers = {5, 2, 8, 1};

for(int i = 0; i < numbers.length - 1; i++) {
    for(int j = 0; j < numbers.length - i - 1; j++) {
        if(numbers[j] > numbers[j + 1]) {
            // Swap
            int temp = numbers[j];
            numbers[j] = numbers[j + 1];
            numbers[j + 1] = temp;
```

```
        }
    }
}

System.out.println("Sorted Array:");
for(int num : numbers) {
    System.out.print(num + " ");
}
```

---

11. Linear Search

```
int search(int[] arr, int key) {
    for(int i = 0; i < arr.length; i++) {
        if(arr[i] == key) return i;
    }
    return -1;
}
```

---

12. Binary Search (Array must be sorted)

```
int binarySearch(int[] arr, int key) {
    int low = 0, high = arr.length - 1;
    while(low <= high) {
        int mid = (low + high) / 2;
        if(arr[mid] == key) return mid;
        else if(arr[mid] < key) low = mid + 1;
        else high = mid - 1;
    }
    return -1;
}
```

---

13. Key OOP Principles Summary

Principle	Keyword	Description
Inheritance	extends	Child class inherits properties of parent.
Abstraction	abstract	Hides implementation details.
Interface	implements	Contract of methods to be defined.

Principle	Keyword	Description
Polymorphism	Method Overriding	Same method name, different behavior.
Encapsulation	private + getters/setters	Hides data, controls access.

---

14. Scanner for Input

```
import java.util.Scanner;
Scanner sc = new Scanner(System.in);
int n = sc.nextInt();
String name = sc.nextLine();
```

---

15. Important Notes

- **Constructors**: Same name as class, no return type.
- **Overriding**: Subclass changes parent class method behavior.
- **Final**: Prevents method overriding or class inheritance.

Quick Tips

- Always close Scanner: `sc.close();`
- `ArrayList` can replace arrays for dynamic sizing.
- Sort arrays manually using Bubble/Selection/Insertion Sort.
- Use enhanced for-loop (`for-each`) for simple traversals.

Prepared for: Java OOP Lab Test Cheat Sheet