Java OOP Cheat Sheet

Exam Reference Summary

October 9, 2025

1. Class, Objects & Constructors

```java
// Class Definition
public class Person {
    private String name;
    private int age;
    public static final double PI =
        3.14159;

    // Constructor
    public Person(String name, int age
        ) {
        this.name = name;
        this.age = age;
    }

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

    // Setter
    public void setAge(int age) { this
        .age = age; }

    // toString()
    public String toString() {
        return name + " (" + age + "
            years)";
    }
}
```

Notes:

- Use `private` for encapsulation.
- `this` refers to current instance.
- `static` belongs to the class, not instances.
- `final` makes a variable constant.

2. Static, Final & Constants

```java
public class MathUtils {
    public static final double PI =
        3.14159;
    public static int add(int a, int b
        ) {
        return a + b;
    }
}
System.out.println(MathUtils.PI);
System.out.println(MathUtils.add(3,4))
    ;
```

Tip: Use uppercase for constants (`MAX_VALUE`).

3. Casting Examples

```java
// Implicit casting (widening)
int i = 10;
double d = i; // OK

// Explicit casting (narrowing)
double x = 9.8;
int n = (int) x; // truncates decimal

// Object casting
Object obj = "Hello";
String str = (String) obj; // OK
```

Tip: Use `instanceof` before casting safely.

4. System.out.printf Formatting

```java
String name = "Alice";
int age = 23;
double balance = 1234.567;

System.out.printf("Name: %s | Age: %d
    | Balance: %.2f euros %n",
                  name, age, balance);

// %s - string
// %d - integer
// %f - floating number
// %.2f - 2 decimal places
// %n - new line
```

5. Collections Overview

ArrayList

```java
ArrayList<String> names = new
    ArrayList<>();
names.add("Alice");
names.add("Bob");
```

```
names.remove("Alice");
System.out.println(names.get(0));
System.out.println(names.size());
```

Complexity: Access O(1), Add/Remove O(n).

HashMap

```
HashMap<String, Integer> ages = new HashMap<>();
ages.put("Alice", 23);
ages.put("Bob", 30);
System.out.println(ages.get("Alice"));
System.out.println(ages.containsKey("Bob"));
```

Note: Keys are unique - adding a duplicate replaces the value. **Complexity:** Avg O(1), Worst O(n).

HashSet

```
HashSet<String> fruits = new HashSet<>();
fruits.add("Apple");
fruits.add("Apple"); // ignored
System.out.println(fruits.size()); // 1
```

Use when you don't want duplicates.

TreeMap

```
import java.util.TreeMap;

TreeMap<String, Integer> scores = new TreeMap<>();
scores.put("Alice", 90);
scores.put("Bob", 85);
scores.put("Charlie", 95);

System.out.println(scores); // Sorted by keys
System.out.println(scores.firstKey()); // Smallest key
System.out.println(scores.lastEntry()); // Largest key
System.out.println(scores.headMap("Charlie")); // Keys < "Charlie"
```

Notes:

- Keys are sorted in natural order or by a custom `Comparator`.
- Cannot have `null` keys.
- Useful for ordered mappings.

Complexity: All main operations O(log n).

6. Loops & Iteration

```
for (int i=0; i<list.size(); i++) {
    System.out.println(list.get(i));
}

for (String s : list) {
    System.out.println(s);
}

Iterator<String> it = list.iterator();
while (it.hasNext()) {
    System.out.println(it.next());
}
```

7. Equality: == vs equals()

- `==` compares object references.
- `equals()` compares logical content.

```
String a = new String("Hello");
String b = new String("Hello");
System.out.println(a == b);        // false
System.out.println(a.equals(b));   // true
```

8. Common Pitfalls

- Arrays use `.length`, Lists use `.size()`.
- `System.out.println()` automatically calls `toString()`.
- `NullPointerException` → uninitialized object.
- Initialize lists before using: `list = new ArrayList<>();`

9. Random Numbers

```
import java.util.Random;
Random r = new Random();
int x = r.nextInt(10); // 0 to 9
double y = Math.random(); // 0.0 to 1.0
```

10. File and String Tricks

```
String s = "Hello World";
System.out.println(s.toUpperCase());
System.out.println(s.substring(0, 5));
System.out.println(s.contains("World"));
```

11. Reading User Input

```java
import java.util.Scanner;

Scanner sc = new Scanner(System.in);

// Read types
int age = sc.nextInt();
double price = sc.nextDouble();
String name = sc.nextLine(); // reads line
String token = sc.next(); // reads one word

System.out.printf("Name: %s, Age: %d%n", name, age);

sc.close();
```

Notes:

- Always close `Scanner` when done.
- Use `nextLine()` carefully after `nextInt()` or `nextDouble()` (consume newline).

12. Generics

```java
// Generic class
public class Box<T> {
    private T value;
    public Box(T value) { this.value = value; }
    public T getValue() { return value; }
    public void setValue(T value) { this.value = value; }
}

// Generic method
public static <E> void printArray(E[] array) {
    for (E elem : array) {
        System.out.println(elem);
    }
}

// Usage
Box<Integer> intBox = new Box<>(42);
Box<String> strBox = new Box<>("Hello");
```

Notes:

- Type parameter between `< >` (e.g., `<T>`).
- Enforces type safety — no need for casting.
- Can use multiple types: `<K, V>` (for maps, pairs, etc.).

13. Plots

```java
import javax.swing.*;
import java.awt.*;

public class SimplePlot extends JPanel {
    protected void paintComponent(Graphics g) {
        super.paintComponent(g);
        g.drawLine(50, 50, 150, 150);
        g.fillOval(100, 100, 10, 10);
    }
    public static void main(String[] args) {
        JFrame f = new JFrame("Plot");
        f.add(new SimplePlot());
        f.setSize(300, 300);
        f.setVisible(true);
    }
}
```

Remarques :

- Redéfinir `paintComponent(Graphics g)`.
- Utiliser `repaint()` pour rafraîchir.
- Couleur : `g.setColor(Color.RED);` avant de dessiner.

14. Arrays

Declaration

```java
int[] array1;
double[] array2;
Position[] array3;
```

Declares references to arrays, but does not allocate memory.

Creation and Initialization

```java
int[] array1 = new int[2];
double[] array2 = new double[4];
Position[] array3 = new Position[3];

int[] array4 = { 1, 2 };
double[] array5 = { 1.1, 2.2, 3.3, 4.4 };
Position[] array6 = {
    new Position(1, 0),
    new Position(0, 1),
    new Position(1, 1)
};
```

Access and Length

```
int x = array4[0];
array4[1] = 10;
System.out.println(array5.length); // prints 4
```

Note: Accessing invalid indices throws `ArrayIndexOutOfBoundsException`. Array size is fixed.

Traversing Arrays

For loop:

```
double[] arr = {1.1, 2.2, 3.3};
for (int i = 0; i < arr.length; i++) {
    System.out.println(arr[i]);
}
```

For-each loop:

```
double sum = 0;
for (double val : arr){sum += val};
System.out.println("sum = " + sum);
```

Display

```
import java.util.Arrays;
double[] arr = {1.1, 2.2, 3.3};
System.out.println(Arrays.toString(arr));
```

Multidimensional arrays:

```
int[][] mat = {{1,2},{3,4}};
System.out.println(Arrays.deepToString(mat));
```

Output: `[[1, 2], [3, 4]]`

15. Enumerations

Basic Enum Definition

```
public enum Day {
    SUNDAY, MONDAY, TUESDAY, WEDNESDAY,
    THURSDAY, FRIDAY, SATURDAY
}
```

Enum constants are written in uppercase. Each constant is an instance of the enum type.

Usage Example

```
public class EnumTest {
    Day day;
    public EnumTest(Day day) { this.day = day; }

    public void tellItLikeItIs() {
        switch (day) {
            case MONDAY -> System.out.println("Mondays are bad.");
            case FRIDAY -> System.out.println("Fridays are better.");
            case SATURDAY, SUNDAY -> System.out.println("Weekends are best.");
            default -> System.out.println("Midweek days are so-so.");
        }
    }

    public static void main(String[] args) {
        new EnumTest(Day.MONDAY).tellItLikeItIs();
    }
}
```

Output: `Mondays are bad.`

Notes:

- Enum constants are created automatically at class load.
- Enum constructors are always private or package-private.
- `values()` returns all constants.
- Enums implicitly extend `java.lang.Enum`.