

Java OOP Cheat Sheet

Exam Reference Summary

October 5, 2025

1. Class, Objects & Constructors

```
1 // Class Definition
2 public class Person {
3     private String name;
4     private int age;
5     public static final double PI =
6         3.14159;
7
8     // Constructor
9     public Person(String name, int age) {
10         this.name = name;
11         this.age = age;
12     }
13
14     // Getter
15     public String getName() { return
16         name; }
17
18     // Setter
19     public void setAge(int age) { this
20         .age = age; }
21
22     // toString()
23     public String toString() {
24         return name + " (" + age + "
25             years)";
26     }
27 }
```

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

```
1 public class MathUtils {
2     public static final double PI =
3         3.14159;
4     public static int add(int a, int b) {
5         return a + b;
6     }
7 }
```

```
6 }
7 System.out.println(MathUtils.PI);
8 System.out.println(MathUtils.add(3,4))
9 ;
```

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

3. Casting Examples

```
1 // Implicit casting (widening)
2 int i = 10;
3 double d = i; // OK
4
5 // Explicit casting (narrowing)
6 double x = 9.8;
7 int n = (int) x; // truncates decimal
8
9 // Object casting
10 Object obj = "Hello";
11 String str = (String) obj; // OK
```

Tip: Use `instanceof` before casting safely.

4. System.out.printf Formatting

```
1 String name = "Alice";
2 int age = 23;
3 double balance = 1234.567;
4
5 System.out.printf("Name: %s | Age: %d
6     | Balance: %.2f euros %n",
7         name, age, balance);
8
9 // %s - string
10 // %d - integer
11 // %f - floating number
12 // %.2f - 2 decimal places
13 // %n - new line
```

5. Collections Overview

ArrayList

```
1 ArrayList<String> names = new
2     ArrayList<>();
3 names.add("Alice");
4 names.add("Bob");
```

```

4 names.remove("Alice");
5 System.out.println(names.get(0));
6 System.out.println(names.size());

```

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

HashMap

```

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

```

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

HashSet

```

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

```

Use when you don't want duplicates.

TreeMap

```

1 import java.util.TreeMap;
2
3 TreeMap<String, Integer> scores = new
    TreeMap<>();
4 scores.put("Alice", 90);
5 scores.put("Bob", 85);
6 scores.put("Charlie", 95);
7
8 System.out.println(scores); // Sorted
    by keys
9 System.out.println(scores.firstKey());
    // Smallest key
10 System.out.println(scores.lastEntry());
    // Largest key
11 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

```

1 for (int i=0; i<list.size(); i++) {
2     System.out.println(list.get(i));
3 }
4
5 for (String s : list) {
6     System.out.println(s);
7 }
8
9 Iterator<String> it = list.iterator();
10 while (it.hasNext()) {
11     System.out.println(it.next());
12 }

```

7. Equality: == vs equals()

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

```

1 String a = new String("Hello");
2 String b = new String("Hello");
3 System.out.println(a == b); //
    false
4 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

```

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

```

10. File and String Tricks

```

1 String s = "Hello World";
2 System.out.println(s.toUpperCase());
3 System.out.println(s.substring(0, 5));
4 System.out.println(s.contains("World")
    );

```

11. Reading User Input

```
1 import java.util.Scanner;
2
3 Scanner sc = new Scanner(System.in);
4
5 // Read types
6 int age = sc.nextInt();
7 double price = sc.nextDouble();
8 String name = sc.nextLine(); // reads
    line
9 String token = sc.next(); // reads one
    word
10
11 System.out.printf("Name: %s, Age: %d%n", name, age);
12
13 sc.close();
```

Notes:

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

12. Generics

```
1 // Generic class
2 public class Box<T> {
3     private T value;
4     public Box(T value) { this.value =
        value; }
5     public T getValue() { return value
        ; }
6     public void setValue(T value) {
        this.value = value; }
7 }
8
9 // Generic method
10 public static <E> void printArray(E[]
    array) {
11     for (E elem : array) {
12         System.out.println(elem);
13     }
```

```
14 }
15
16 // Usage
17 Box<Integer> intBox = new Box<>(42);
18 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

```
1 import javax.swing.*;
2 import java.awt.*;
3
4 public class SimplePlot extends JPanel
    {
5     protected void paintComponent(
        Graphics g) {
6         super.paintComponent(g);
7         g.drawLine(50, 50, 150, 150);
8         g.fillOval(100, 100, 10, 10);
9     }
10    public static void main(String[]
        args) {
11        JFrame f = new JFrame("Plot");
12        f.add(new SimplePlot());
13        f.setSize(300, 300);
14        f.setVisible(true);
15    }
16 }
```

Remarques :

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