FREE CHEAT SHEET

Java OOP Quick Reference, for Singapore CS Students

Two pages. The OOP patterns NUS CS2030S, NTU CZ2002, and SMU IS200 actually grade for, plus the bugs that quietly cost marks.

CLASS BASICS

Constructor and `this`

```
public class Student {
    private String name;
    private int id;

    public Student(String name, int id) {
        this.name = name;
        this.id = id;
    }
}
```

Getter / setter

```
public String getName() {
    return name;
}

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

INHERITANCE AND POLYMORPHISM

extends, super, override

```
public class Tutor extends Person {
    private String subject;

    public Tutor(String n, String s) {
        super(n);
        this.subject = s;
    }

    @Override
    public String describe() {
        return super.describe()
            + ", teaches " + subject;
    }
}
```

Interface vs abstract class

```
// abstract: shared state + behaviour
public abstract class Shape {
    protected String name;
    public abstract double area();
}

// interface: pure contract
public interface Drawable {
    void draw(Graphics g);
}
```

Pick interface for "what something *can do*". Pick abstract for "what something *is*" with shared code.

GENERICS, FAST

```
// type parameter
public class Box<T> {
    private T value;
```

```
    public Box(T v) { value = v; }
    public T get() { return value; }
}

// PECS, Producer Extends Consumer Super
void copy(List<? extends Number> src,   // reads from src, producer
          List<? super Integer> dst) { // writes to dst, consumer
    for (Number n : src) dst.add((Integer) n);
}
```

FIVE BUGS THAT COST MARKS

1. Forgetting equals/hashCode

If you override one, override both. HashSet and HashMap break silently otherwise.

```
@Override
public boolean equals(Object o) {
    if (this == o) return true;
    if (!(o instanceof Student)) return false;
    return ((Student) o).id == this.id;
}

@Override
public int hashCode() { return Integer.hashCode(id); }
```

2. Mutating a field exposed via getter

Returning the internal list lets callers mutate it. Wrap in unmodifiable or copy.

```
// leak
public List<Course> getCourses() { return courses; }

// safe
public List<Course> getCourses() {
    return Collections.unmodifiableList(courses);
}
```

3. NullPointerException on chained calls

Use `Optional` to make absence explicit.

```
String city = student.getAddress().getCity(); // NPE risk

String city = Optional.ofNullable(student)
    .map(Student::getAddress)
    .map(Address::getCity)
    .orElse("Unknown");
```

4. Mutable default state in static fields

Static collections are shared across all instances. Test cases reusing a class hit ghost data.

```java
public class Course {
    private static List<Student> roster = new ArrayList<>();
    // every Course shares the same roster, almost certainly a bug
}
```

5. instanceof + casting instead of polymorphism

If you find yourself writing a chain of `instanceof`, you're working against OOP. Override a method instead.

STREAMS AND LAMBDAS, ONE-LINERS

```java
List<String> names = students.stream()
    .filter(s -> s.getGpa() > 4.0)
    .map(Student::getName)
    .sorted()
    .toList();

int total = orders.stream().mapToInt(Order::getQty).sum();
Map<String, Long> byCity = users.stream().collect(
    Collectors.groupingBy(User::getCity, Collectors.counting()));
```

MODULES WHERE THIS SAVES YOU THE MOST MARKS

NUS CS2030S (functional + OOP, generics-heavy), NUS CS2103T (software engineering team project), NTU CZ2002 (OOP design), SMU IS200, SIT INF1003.