S7 Inf A3 — Java for Graphical and Mobile Programming · Centrale Lyon

Stéphane Derrode

CM2

Collections, Exceptions & File I/O

- ArrayList and HashMap — the two workhorses
- Stack — LIFO data structure
- Exception handling — try / catch / finally / throws
- Custom exception classes
- File I/O — reading and writing text files

Part 1 — Java Fundamentals

2h lecture

Module Overview

Part 1 — Java Fundamentals	Part 2 — GUI with Swing	Part 3 — Android
CM1 From Python to Java	**CM4** Swing: Components & Layouts	**CM6** Android Architecture
CM2 Collections, Exceptions & I/O	**CM5** Graphics 2D & Events	**CM7** Layouts, Navigation & Intents
CM3 Inheritance, Interfaces		**CM8** Persistence & Threads

Projects: ★ Spider Game (Part 2) ★ Calculator / Currency Converter (Part 3)

Exam 50% + BEs 50%

Stéphane Derrode — Centrale Lyon · S7 Inf A3

CM2 — Agenda

01 Collections framework overview

02 ArrayList<T>

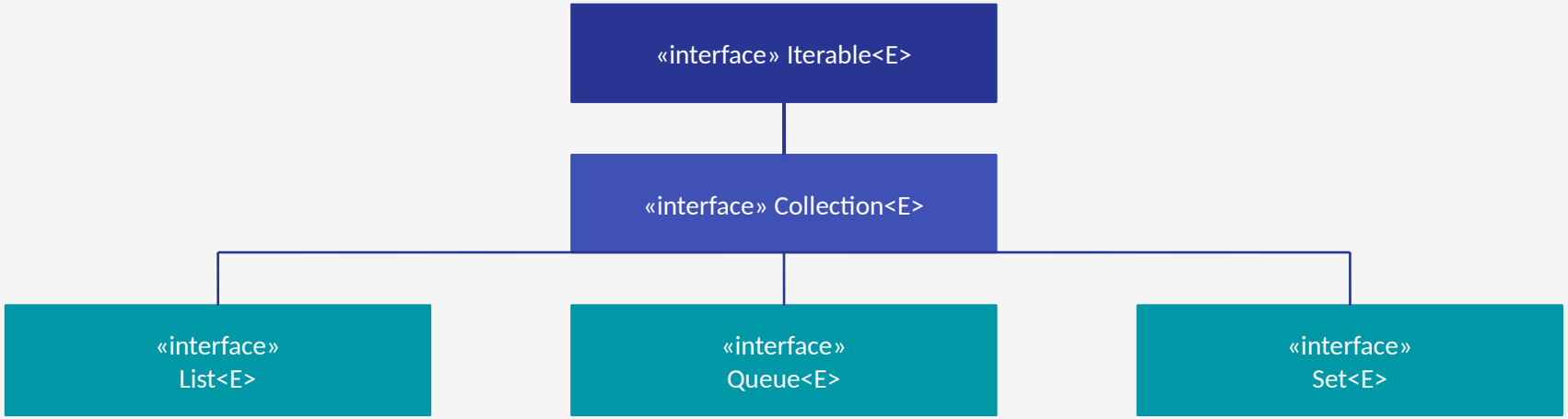
03 HashMap<K,V>

04 Stack<T>

05 Exception handling

06 Custom exceptions & File I/O

The Java Collections Framework



ArrayList<E>

resizable array

★ covered in this CM

LinkedList<E>

doubly linked

Stack<E>

LIFO

HashMap<K,V>

key → value

★ covered in this CM

Programming to interfaces

Always declare with the interface type: List<String> names = new ArrayList<>(); — this makes it easy to swap implementations later.

ArrayList<T> — Dynamic Arrays

Python list → Java ArrayList

```
import java.util.ArrayList;
import java.util.List;
import java.util.Collections;

// Declaration — use the interface type
List<Voiture> fleet = new ArrayList<>();

// Adding elements
fleet.add(new Voiture(4));
fleet.add(new Voiture(6));
fleet.add(new Voiture(28));

// Accessing
Voiture v = fleet.get(0);        // Voiture(4)
System.out.println(fleet.size()); // 3

// Iterating (for-each)
for (Voiture car : fleet) {
    car.demarre();
    System.out.println(car);
}

// Contains, remove, sort
fleet.contains(v);               // true
```

Generic type <T>

ArrayList<Voiture> only holds Voiture objects. Type safety is checked at compile time — no accidental mixing.

Primitives need wrappers

ArrayList<int> is INVALID. Use ArrayList<Integer>. Java auto-boxes int ↔ Integer transparently.

Key methods

add(e) add(i,e) get(i) set(i,e) remove(i) size() isEmpty() contains(e) clear() Collections.sort(list)

Python comparison

Python: fleet.append(v) fleet[0] len(fleet) for v in fleet: — same concepts, different syntax.

HashMap<K,V> — Key/Value Storage

Python dict → Java HashMap

```
import java.util.HashMap;
import java.util.Map;

// Map: owner name → their Voiture
Map<String, Voiture> garage = new HashMap<>();

// Adding entries
garage.put("Alice", new Voiture(4));
garage.put("Bob",   new Voiture(6));

// Accessing
Voiture v = garage.get("Alice");      // Voiture(4)
boolean has = garage.containsKey("Bob"); // true

// Safe get — avoids NullPointerException
Voiture val = garage.getOrDefault(
    "Dave", new Voiture(5));          // default

// Iterating over entries
for (Map.Entry<String,Voiture> e
        : garage.entrySet()) {
    System.out.println(
        e.getKey() + " → " + e.getValue());
}
```

Key → Value

⚠ get() can return null

If the key doesn't exist, get() returns null. Calling any method on null throws NullPointerException. Always use getOrDefault() or containsKey() first.

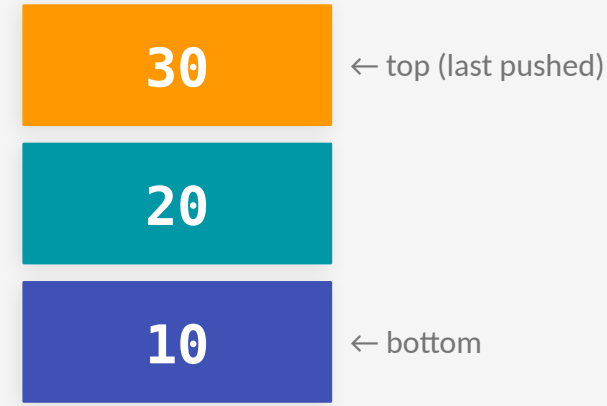
Python comparison

garage["Alice"] garage.get("Dave", default) for k,v in garage.items() — same concepts, different syntax.

Stack<T> — Last In, First Out

```
import java.util.Stack;

Stack<Integer> stack = new Stack<>();

// Push elements
stack.push(10);
stack.push(20);
stack.push(30);

System.out.println(stack.peek()); // 30 (no
remove)
System.out.println(stack.pop());  // 30
(removes it)
System.out.println(stack.pop());  // 20
System.out.println(stack.size()); // 1

// Always check before popping!
if (!stack.isEmpty()) {
    int top = stack.pop();
}
```

LIFO — Last In, First Out

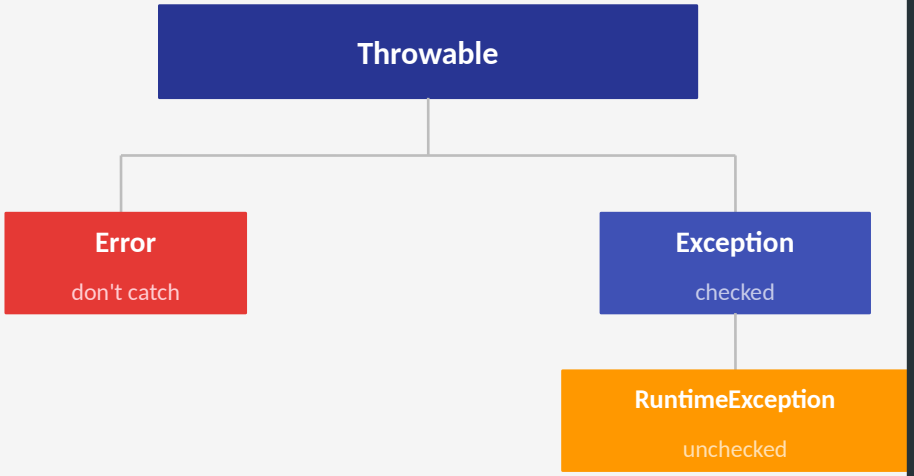


Typical use cases

- Expression / RPN evaluation
- Undo / redo in editors
- Backtracking algorithms
- The JVM call stack itself

Exception Handling — Hierarchy & Basics

Exception hierarchy



Common runtime exceptions

`NullPointerException`	Method called on null reference
`ArrayIndexOutOfBoundsException`	Index < 0 or >= length
`IllegalArgumentException`	Invalid argument value
`ArithmeticException`	e.g. division by zero
`NumberFormatException`	Bad string-to-number parse

```
try {
    Voiture v = fleet.get(10); // may throw
    v.demarre();               // NPE if null

} catch (IndexOutOfBoundsException e) {
    System.out.println("Index error: "
        + e.getMessage());

} catch (NullPointerException e) {
    System.out.println("Null: "
        + e.getMessage());

} catch (Exception e) {   // catch-all (last)
    System.out.println("Unexpected: "
        + e.getMessage());

} finally {
    // always runs — even if exception thrown
    // use for cleanup: close files, etc.
    System.out.println("Done.");
}

// Multi-catch (Java 7+) — same handler
try {
    riskyOperation();
} catch (IOException | NumberFormatException e) {
    System.out.println("Caught: " + e);
}
```

Checked vs Unchecked — When to Use Each

Checked Exception

```
// Compiler FORCES you to handle it
public void readFile(String path)
        throws IOException {
    BufferedReader br =
        new BufferedReader(
            new FileReader(path));
    // ...
}

// Caller MUST handle it:
try {
    readFile("data.txt");
} catch (IOException e) {
    System.out.println("Error: "
        + e.getMessage());
}
```

Use Checked when...
The caller can reasonably recover — file not found, network error, database unavailable.

Unchecked Exception

```
// Extends RuntimeException
// Compiler does NOT force handling
public void setPuissance(int p) {
    if (p <= 0)
        throw new IllegalArgumentException(
            "Power must be > 0: " + p);
    this.puissance = p;
}

// Caller may or may not catch:
try {
    v.setPuissance(-5);
} catch (IllegalArgumentException e) {
    System.out.println(e.getMessage());
}
```

Use Unchecked when...
The error signals a programming mistake — invalid argument, null pointer, index out of bounds. Caller usually cannot recover.

Custom Exception Classes

```
// 1. Define the exception (from your slides)
public class InsufficientFundsException
        extends Exception {      // checked!

    private double amount;       // extra data

    public InsufficientFundsException(double a) {
        super("Insufficient funds: need " + a);
        this.amount = a;
    }
    public double getAmount() { return amount; }
}
```

```
// 2. Throw it in your class
public class CheckingAccount {
    private double balance;
    public void deposit(double amount) {
        balance += amount;
    }
    public void withdraw(double amount)
            throws InsufficientFundsException {
        if (amount <= balance) {
            balance -= amount;
        } else {
            throw new InsufficientFundsException(
                amount - balance);
        }
    }
}
```

```
// 3. Catch it specifically
CheckingAccount account = new CheckingAccount();
account.deposit(500.0);

try {
    account.withdraw(100.0);   // OK
    account.withdraw(600.0);   // throws!
} catch (InsufficientFundsException e) {
    System.out.println("Sorry: $"
        + e.getAmount() + " short");
}
```

Rule

File I/O — Reading & Writing Text Files

Writing

```
import java.io.*;

// try-with-resources: auto-closes writer
try (FileWriter fw = new FileWriter("fleet.txt");
     BufferedWriter bw = new BufferedWriter(fw)) {

    for (Voiture v : fleet) {
        bw.write(v.toString());
        bw.newLine();
    }

} catch (IOException e) {
    System.err.println("Write error: "
        + e.getMessage());
}
```

Reading

```
import java.io.*;

try (BufferedReader br =
        new BufferedReader(
            new FileReader("fleet.txt"))) {

    String line;
    while ((line = br.readLine()) != null) {
        System.out.println(line);
    }

} catch (FileNotFoundException e) {
    System.err.println("Not found");
} catch (IOException e) {
    System.err.println("Read error: "
        + e.getMessage());
}
```

try-with-resources (Java 7+)
Any object in try(…) that implements AutoCloseable is automatically closed when the block exits — even if an exception is thrown. This replaces the verbose try/finally pattern and prevents resource leaks. Always use it for files, streams, and database connections.

Relative path
"fleet.txt" is relative to the project root with Maven.

Append mode
new FileWriter("fleet.txt", true) — second arg true = append.

Putting It All Together — A Fleet Manager

```
import java.util.*;
import java.io.*;

public class FleetManager {
    private Map<String, Voiture> garage = new HashMap<>();

    public void register(String owner, Voiture v) {
        garage.put(owner, v);
    }

    public Voiture lookup(String owner) {
        if (!garage.containsKey(owner))
            throw new IllegalArgumentException(
                "Unknown owner: " + owner);
        return garage.get(owner);
    }

    public void saveToFile(String path)
            throws IOException {
        try (BufferedWriter bw =
                new BufferedWriter(new FileWriter(path))) {
            for (Map.Entry<String,Voiture> e
                    : garage.entrySet()) {
                bw.write(e.getKey() + ","
                    + e.getValue().deQuellePuissance());
                bw.newLine();
            }
        }
```

Concepts used

`HashMap<K,V>`
owner → Voiture

`containsKey()`
null-safe lookup

`IllegalArgumentException`
unchecked custom error

`Map.Entry<K,V>`
iteration pattern

`throws IOException`
checked exception

`try-with-resources`
auto-close file

Collections & Exceptions Quick Reference

Structure	Declaration	Key operations	Python equiv.
`ArrayList<T>`	`List<T> l = new ArrayList<>()`	add, get(i), size, remove, contains, sort	`list`
`HashMap<K,V>`	`Map<K,V> m = new HashMap<>()`	put, get, containsKey, remove, entrySet, getOrDefault	`dict`
`Stack<T>`	`Stack<T> s = new Stack<>()`	push, pop, peek, isEmpty, size	`list (as stack)`

Exception cheat sheet

Exception	Cause	Type
`NullPointerException`	Method called on null reference	Unchecked
`IndexOutOfBoundsException`	Index < 0 or >= array/list size	Unchecked
`IllegalArgumentException`	Invalid value passed to method	Unchecked
`IOException`	File not found / read-write failure	**CHECKED ← must catch**

CM2 — Key Takeaways

ArrayList	Dynamic array — add, get, remove, for-each; use List<T> as declared type
HashMap	Key/value — put, get, getOrDefault; iterate with entrySet(); null if key missing
Stack	LIFO — push, pop, peek; always isEmpty() before pop()
Exceptions	try/catch/finally; multi-catch; checked (must handle) vs unchecked (optional)
Custom exc.	extend Exception (checked) or RuntimeException (unchecked); add fields for data
File I/O	try-with-resources for auto-close; BufferedReader/Writer for text files

Up next → TD2: FleetManager — HashMap, RPN calculator (Stack), file persistence