

Exception Handling

try, catch, finally, throw, throws, checked vs unchecked, custom exceptions

Lecture + Lab 4

Branch: week5-exceptions

Learning outcomes 3 and 1

What you will be able to do after today

- Explain what an exception is and what happens when one is not handled.
- Use `try`, `catch`, `finally`, `throw` and `throws`.
- Explain **checked vs unchecked** exceptions and choose between them.
- Write your own exception classes for the LMS.

1. Printing an error is not handling it

In Week 4 the library did this when a book was already out:

```
System.out.println(title + " is already on loan.");
```

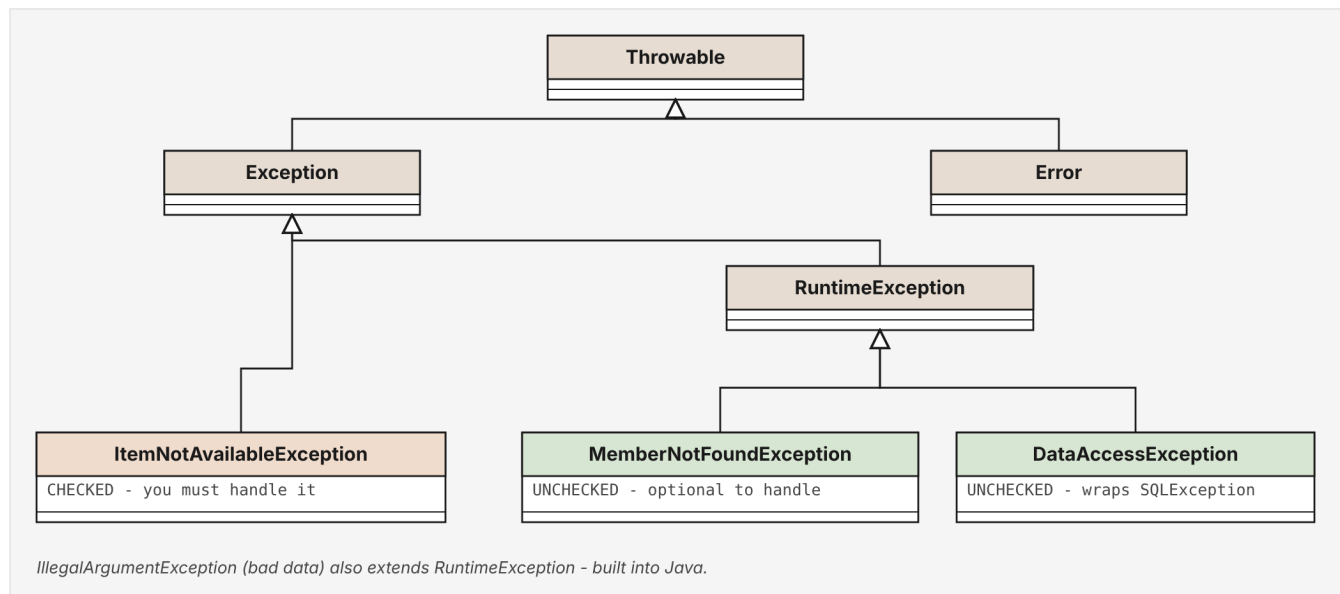
The *caller* has no idea anything went wrong. A GUI cannot show a dialog, a database layer cannot skip the insert. Errors must be **signalled**, so that the caller decides what to do.

2. The mechanics

```
try {
    library.borrowItem("B001", "M002");           // might fail
} catch (ItemNotAvailableException e) {
    System.out.println("Cannot borrow: " + e.getMessage());
} finally {
    System.out.println("attempt finished");        // runs whether or not it failed
}
```

- `throw new X("message")` **creates and throws** an exception. The method stops immediately.
- `throws X` in a method signature **declares** that the method may throw it.
- The exception travels **up the call stack** until some `catch` matches. If none does, the program crashes with a stack trace.
- Several `catch` blocks are checked **in order**; put specific types before general ones.

3. Checked vs unchecked



	Checked	Unchecked
Extends	Exception	RuntimeException
Compiler forces handling?	Yes (catch or throws)	No
Use for	Problems a caller can reasonably recover from	Programming errors and bad data
LMS example	ItemNotAvailableException	MemberNotFoundException, IllegalArgumentException

Tip Ask: "Can the caller do something sensible about it?" An item already on loan: yes, tell the user and pick another. A member id that does not exist: usually a bug or bad data.

4. Your own exceptions

```
public class ItemNotAvailableException extends Exception { // CHECKED
    public ItemNotAvailableException(String message) { super(message); }
}

public class MemberNotFoundException extends RuntimeException { // UNCHECKED
    private final String memberId;
    public MemberNotFoundException(String memberId) {
        super("No member with id '" + memberId + "'");
        this.memberId = memberId;
    }
    public String getMemberId() { return memberId; }
}
```

5. The LMS after Week 5

`loan()` now **throws** instead of printing, and the interface says so:

```

public interface Loanable {
    void loan(Member member) throws ItemNotAvailableException;
    void returnItem() throws ItemNotAvailableException;
}

@Override
public void loan(Member member) throws ItemNotAvailableException {
    if (isBorrowed) throw new ItemNotAvailableException(title + " is already on loan.");
    if (member.getBorrowedCount() >= member.getMaxBorrowLimit())
        throw new ItemNotAvailableException(member.getName() + " has reached the limit of " +
member.getMaxBorrowLimit() + " items.");
    isBorrowed = true;
    borrower = member;
    member.incrementBorrowedCount();
}

```

And `Library.borrowItem(itemId, memberId)` is the single entry point the GUI will call later:

```

public void borrowItem(String itemId, String memberId) throws ItemNotAvailableException {
    Member member = findMember(memberId); // may throw MemberNotFoundException (unchecked)
    LibraryItem item = findItem(itemId); // may throw ItemNotAvailableException (checked)
    item.loan(member);
}

```

Demo output

```

--- 2. Already on loan (checked exception) ---
Cannot borrow: Clean Code is already on loan.
(attempt finished: item B001, member M002)
--- 3. Unknown member (unchecked exception) ---
Unknown member: M999
(attempt finished: item D001, member M999)
--- 5. Returning something not on loan ---
Cannot return: Introduction to Algorithms is not on loan.
--- 7. Student limit of 3 ---
Cannot borrow: Nadeesha Perera has reached the limit of 3 items.

```

6. Good habits

Do	Don't
Catch the most specific exception you can handle	<code>catch (Exception e) {}</code> (swallowing everything)
Give messages that say what and which value	"Error"
Handle errors where you can do something about them	Catch and print in every layer
Use <code>finally</code> (or try-with-resources) to release resources	Leave files or connections open
Keep the cause when wrapping (<code>new X(msg, cause)</code>)	Lose the original stack trace

7. Lab 4 preview (Week 5)

Add `lms.exception` with the two custom exceptions; change `loan()` and `returnItem()` to throw; add `findMember`, `findItem`, `borrowItem`; write a `Main` with at least **six** scenarios, each with `try/catch/finally`. Also fix a real bug: returning an item must free the member's slot. Branch `week5-exceptions`.

8. Self-check

1. What happens to a method that throws an exception nobody catches?
2. Why is `MemberNotFoundException` unchecked but `ItemNotAvailableException` checked? Do you agree?
3. What does `finally` guarantee?
4. Fix this: `catch (Exception e) { } catch (ItemNotAvailableException e) { }`. Why does it not compile?