Day_10_OOPJ_Sanket_Shalukar

Monday, September 08, 2025 11:05 AM

Topics are in the Day_10

1. String
2. Error
3. Exception
4. Exception Handling
5. Types of Exception

String Methods:

- **indexOf(str):** Returns index of first occurrence.
- **toLowerCase():** Converts string to lowercase.
- **toUpperCase():** Converts string to uppercase.
- **trim():** Removes leading and trailing spaces.
- **replace(old, new):** Replaces a character or sequence with another.
- **reverse():** Reverses the string.

Example string:
`Java Programming is interesting.`

(Note: `reverse()` does not directly work on `String`, but works on `StringBuilder` or `StringBuffer`.)

String Builder: (Mutable & Fast)

- A **mutable sequence** of characters (unlike String which is immutable).
- Uses **heap memory** for storage.
- **Performance:** Faster than the `String` class because it does not create new objects on every modification.
- Best choice **when we frequently modify strings** in a single-threaded environment.

Additional Points on StringBuilder:

- **append():** Adds text at the end.
- **insert():** Inserts text at a specified position.
- **delete(start, end):** Removes characters between the given range.
- **replace(start, end, str):** Replaces characters in a given range with another string.
- **reverse():** Reverses the sequence of characters.
- **capacity():** Returns current capacity of the builder (default 16, increases as needed).
- **ensureCapacity(minCapacity):** Ensures the capacity is at least the given minimum.

Difference between String, StringBuffer, and StringBuilder:

- **String:** Immutable, slower for modifications, thread-safe by default.
- **StringBuffer:** Mutable, thread-safe (synchronized), slightly slower than StringBuilder.
- **StringBuilder:** Mutable, not thread-safe, fastest for single-threaded operations.

String Buffer

- Uses **heap memory** for storage.
- **Performance:** Slower than `StringBuilder`.
- Best suited **when we need to modify strings frequently in a multi-threaded environment**.
- **Thread-safe** (synchronized).
- Mutable string (can be changed without creating a new object).

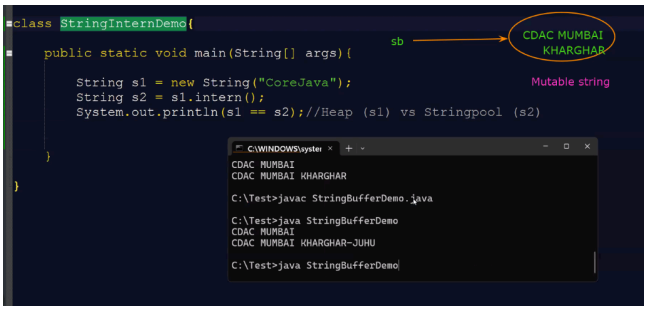
intern() Method

- The `intern()` method in Java ensures that **strings with the same content share a single memory reference** in the **String Pool**.
- When `intern()` is called on a string, it checks the **String Pool**:
- If the pool already contains a string with the same value, it returns the reference from the pool.
- If not, the string is added to the pool and that reference is returned.
- Helps in **memory optimization** and ensures that string literals with the same content point to the same object.

String Pool

- The **String Pool** (or **intern pool**) is a special memory region inside the **Java heap**.

- It stores all string literals and interned strings.
- **Example:**
- `"Java"` stored in the pool → reused whenever `"Java"` appears again.
- But `new String("Java")` creates a new object in the heap, not in the pool (unless `intern()` is called).
- **Benefits:** Saves memory and allows fast string comparisons using == because references are the same.



What is Exception Handling?

- **Exception:** An **unexpected event** that disrupts the normal flow of a program (e.g., divide by zero, file not found).
- **Exception Handling:** A mechanism in Java to handle runtime errors gracefully, so the program does not crash abruptly.

🔹 Key Terms

- **Error:** Serious issues (like OutOfMemoryError) that usually cannot be handled.
- **Exception:** Problems that can be handled by code.
- **Checked Exceptions:** Checked at compile time (e.g., IOException, SQLException).
- **Unchecked Exceptions:** Occur at runtime (e.g., ArithmeticException, NullPointerException).

🔹 Exception Handling Keywords

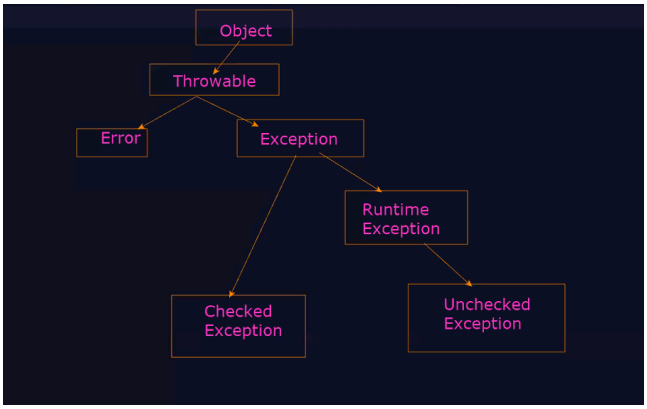
6. **try** → Block of code where exception may occur.
7. **catch** → Handles the exception.
8. **finally** → Always executes (used for cleanup, like closing files).
9. **throw** → Used to explicitly throw an exception.
10. **throws** → Declares exceptions a method can throw.

🔹 Exception Hierarchy

- **Throwable** (base class)
- **Error** (serious issues, cannot be recovered)
- **Exception**
- **Checked Exceptions** (must be handled)
- **Unchecked Exceptions (RuntimeException)**

🔹 Common Exceptions in Java

- **ArithmeticException** → Division by zero.
- **NullPointerException** → Accessing an object with null reference.
- **ArrayIndexOutOfBoundsException** → Accessing array beyond valid index.
- **NumberFormatException** → Invalid conversion from string to number.
- **IOException** → Issues with input/output operations.



Difference Between Exception and Error

Exception

- An issue in a program that prevents the normal flow of execution is known as an **Exception**.
- Occurs because of issues in the program (logical/runtime mistakes).
- **Recoverable** → Can be handled using code.
- Classified into **Checked** and **Unchecked** exceptions.
- Can be handled using keywords → `try`, `catch`, `finally`, `throw`, `throws`.

  Examples:
  - `ArithmeticException`
  - `ClassCastException`
  - `NullPointerException`
  - `ClassNotFoundException`

Error

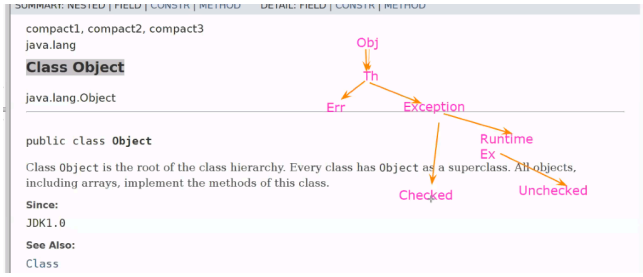
- Identification of an unexpected condition that occurs due to **lack of system resources** is known as an **Error**.
- Occurs mainly because of system-level problems (not the programmer's fault).
- **Unrecoverable** → Cannot be handled by the programmer.
- Classified as **Unchecked Errors**.
- No way to handle errors using code.

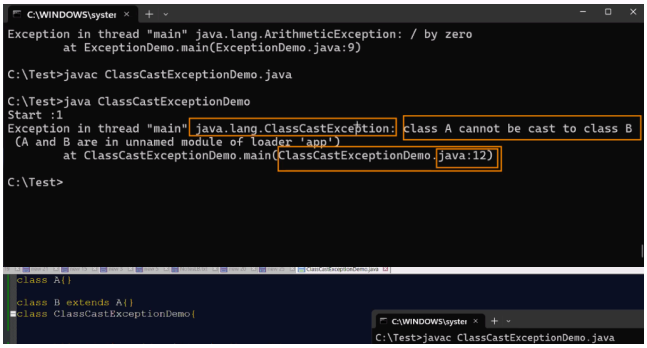
  Examples:
  - `AssertionError`
  - `LinkageError`
  - `VirtualMachineError`
  - `StackOverflowError`

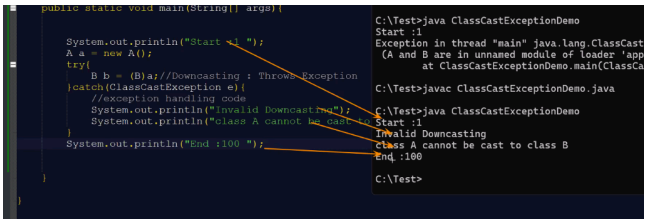
Stack Over flowError

A **Stack Overflow Error** occurs when the program's **call stack memory** is exhausted.

- Commonly caused by:
- Infinite recursion (a method calling itself without an exit condition).
- Deeply nested method calls.
- Example:
  `public void recursive() { recursive(); // infinite recursion → StackOverflowError }`
- It is an **Error**, not an Exception → meaning it **cannot be handled** and usually crashes the program.







What is `printStackTrace()`?

- A method of the **Throwable** class (superclass of Exception & Error).
- Prints the **complete exception details** to the **standard error stream (stderr)**.
- Includes:
  1. **Exception type** → e.g., `java.lang.ClassCastException`
  2. **Description/message** → e.g., `class A cannot be cast to class B`
  3. **Stack trace** → sequence of method calls showing where the exception originated (with line numbers).

```java
public static void main(String[] args){


    System.out.println("Start :1 ");
    A a = new A();
    //try-catch
    //try-catch-finally
    try{
        B b = (B)a;//Downcasting : Throws Exception
    }catch(ClassCastException e){
        System.out.println(e.getMessage());
        System.out.println();
        e.printStackTrace();
        System.out.println();

        //exception handling code
        System.out.println("Invalid Downcasting");
        System.out.println("class A cannot be cast to class B");
    }
    System.out.println("End :100 ");
```

Example in Your Screenshot

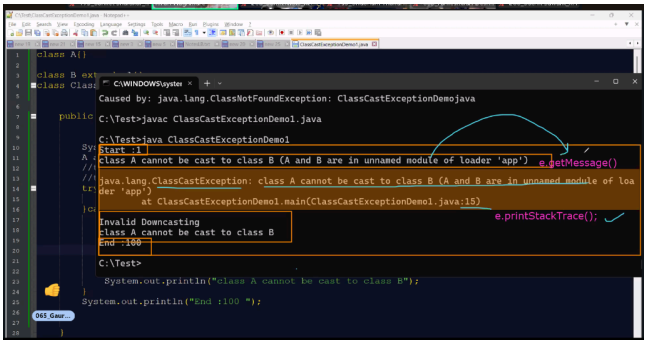
When `e.printStackTrace();` is called:

`java.lang.ClassCastException: class A cannot be cast to class B at ClassCastExceptionDemo1.main(ClassCastExceptionDemo1.java:15)`

- **`java.lang.ClassCastException`** → Exception type.
- **`class A cannot be cast to class B`** → Error message.
- **`at ClassCastExceptionDemo1.main... (line 15)`** → Shows **where in the code** the error happened.

Why we use `printStackTrace()`?

- Helps **debug** by showing exact location of the problem.
- Unlike `getMessage()`, it gives **full details**, not just the message.
- Commonly used in development/logging to find the **root cause**.




```
Types of Exception:
-----------------
1. Checked Exception: Compile-time Exception
-Must be handled using try-catch or declared using throws
-Ex: IOException, ClassNotFoundException

2. Unchecked Exception: Runtime Exceptions
-Do not require mandatory handling
-Ex: ArithmetheticException, ClassCastException
```