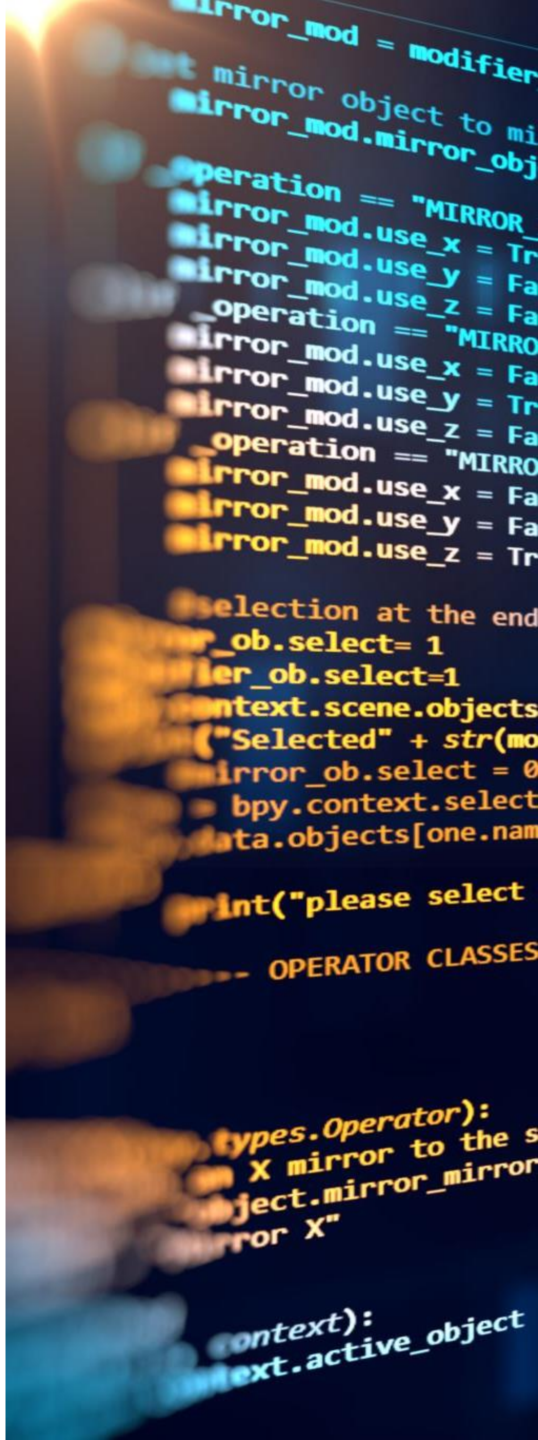
Please use the following QR code to check in and record your attendance.

CS 1027
Fundamentals of Computer Science II

Debugging and Testing

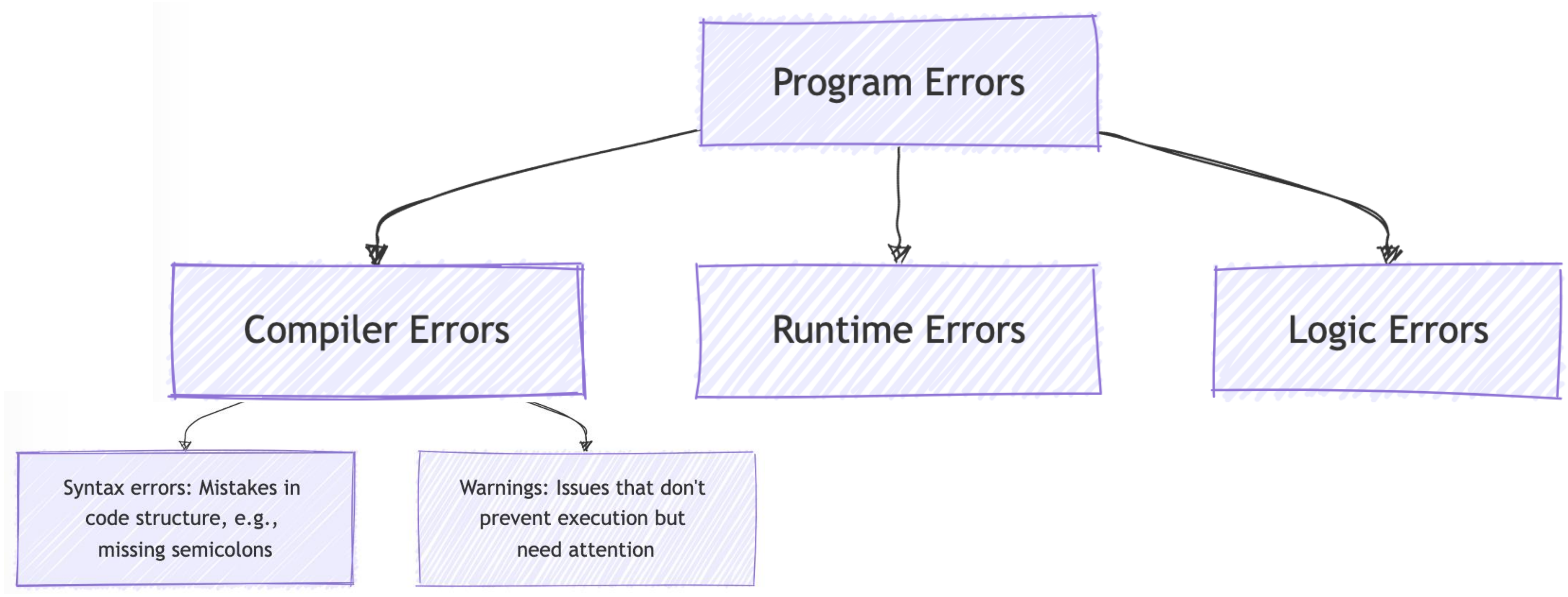
Ahmed Ibrahim



Debugging

Understanding and Fixing Program Errors

Recall: Testing and Debugging


Program Errors
Compiler Errors
Runtime Errors
Logic Errors
Syntax errors: Mistakes in code structure, e.g., missing semicolons
Warnings: Issues that don't prevent execution but need attention

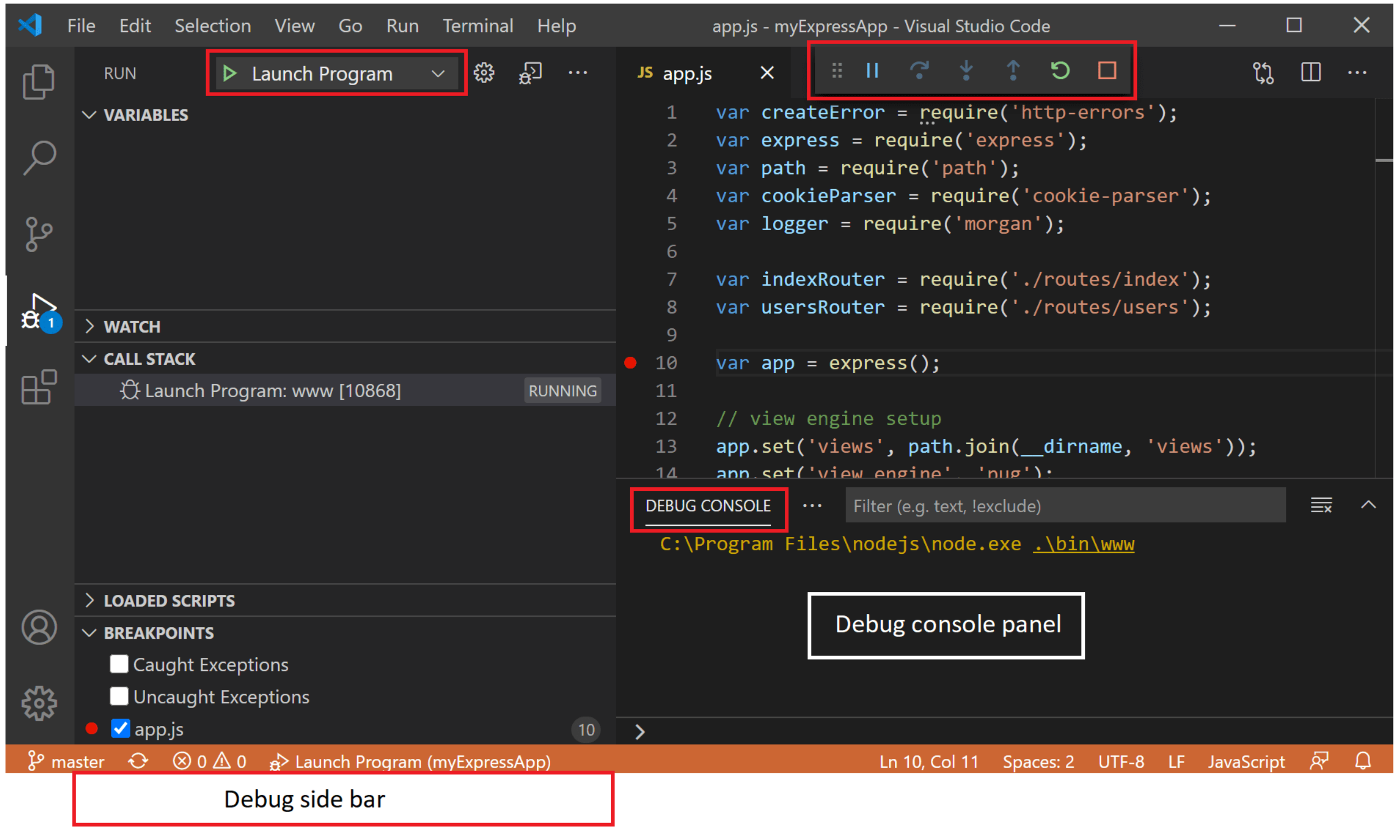
Recall:
Why Are Compiler Errors Confusing?

- **Error Location** – The compiler might point to a line that is different from where the actual error is.
- **Unclear Messages** – Sometimes, the error messages are unclear and might require careful interpretation to understand the root cause.

Using the IDE Debugger

Start debugging

Pause, step over, step in/out, restart, stop



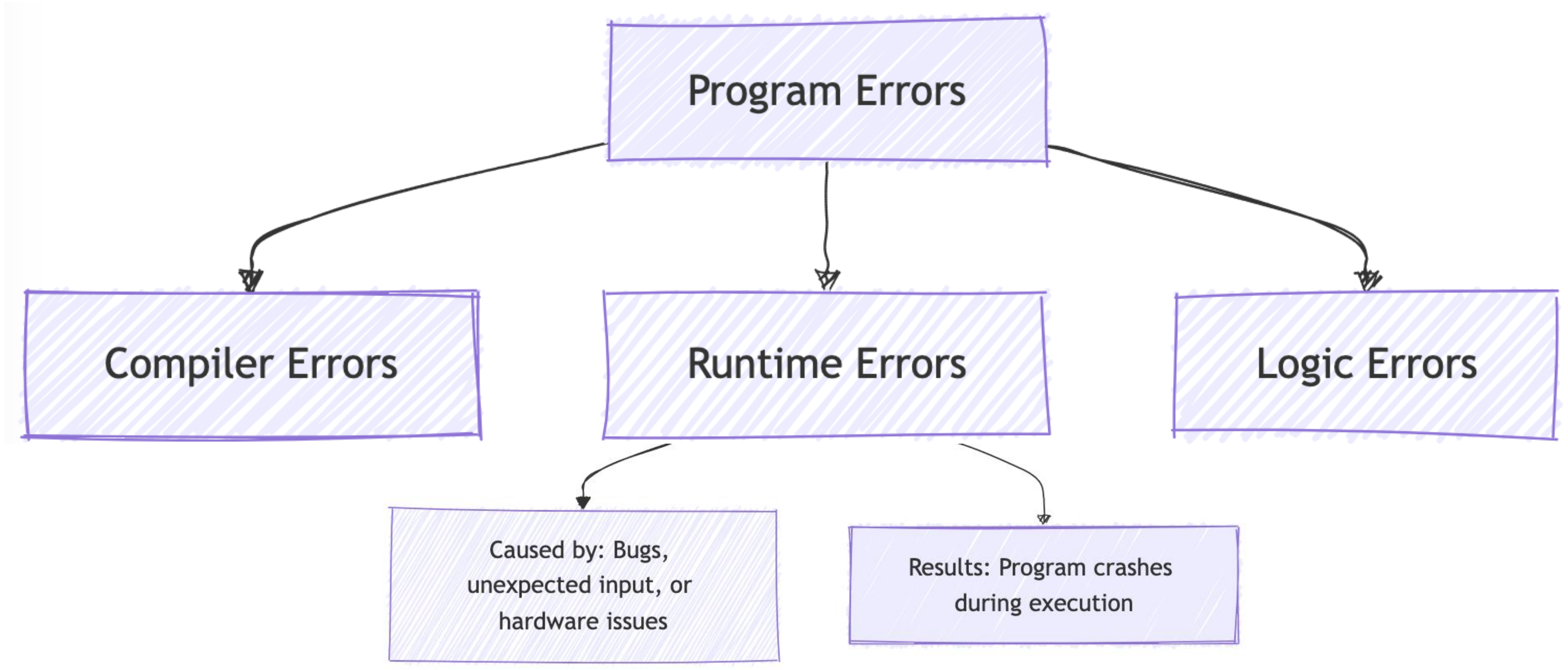
```
var createError = require('http-errors');
var express = require('express');
var path = require('path');
var cookieParser = require('cookie-parser');
var logger = require('morgan');

var indexRouter = require('./routes/index');
var usersRouter = require('./routes/users');

var app = express();

// view engine setup
app.set('views', path.join(__dirname, 'views'));
app.set('view engine', 'pug');
```

```
C:\Program Files\nodejs\node.exe .\bin\www
```

Tips for Troubleshooting Compiler Errors

- Read error messages carefully: They often indicate the line number and type of issue.
- Fix errors one by one: Resolving one error might eliminate others.
- Check for missing or extra symbols: Pay attention to semicolons, brackets, and parentheses.

Testing and Debugging (cont.)

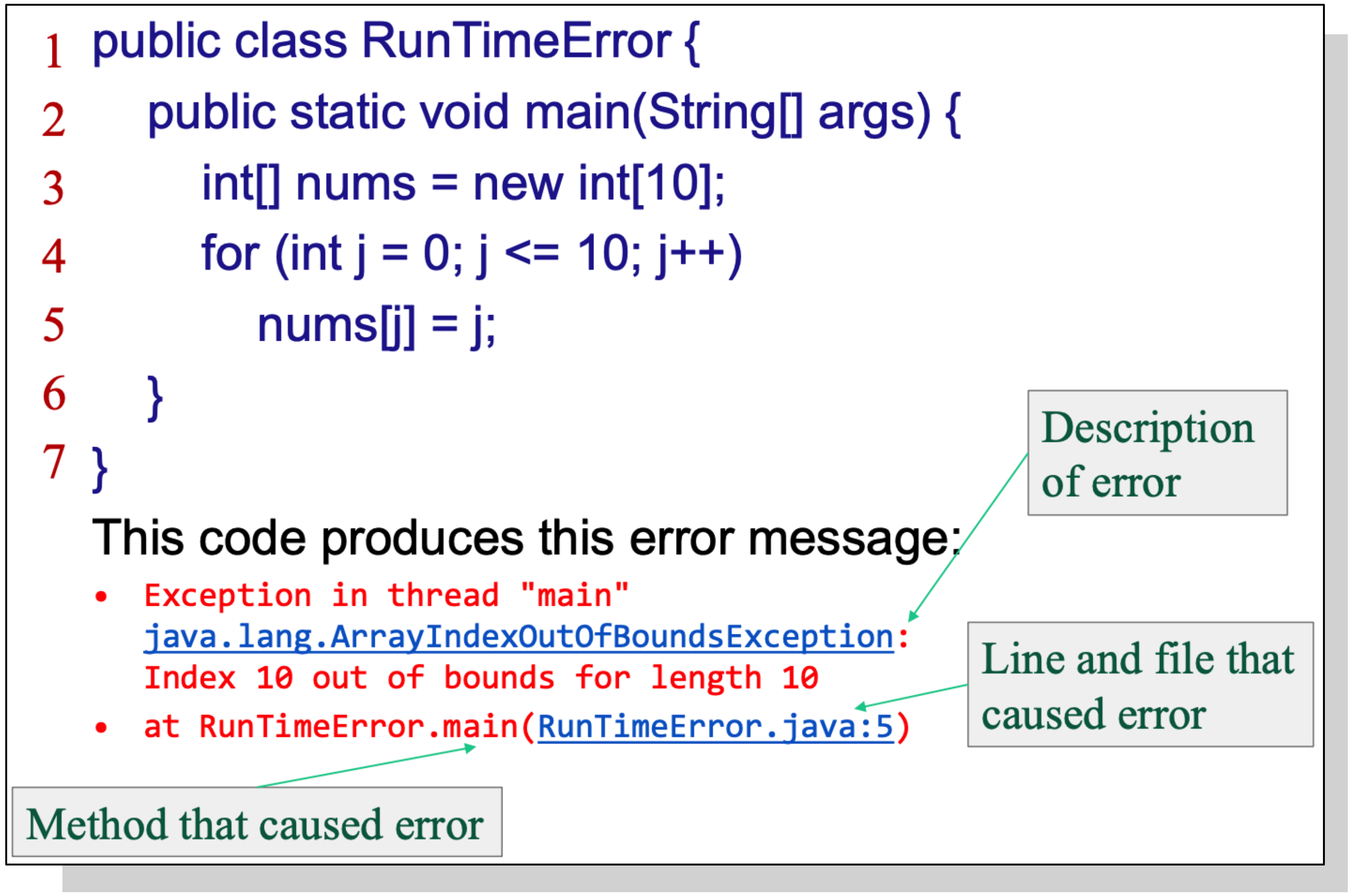

Program Errors
Compiler Errors
Runtime Errors
Logic Errors
Caused by: Bugs, unexpected input, or hardware issues
Results: Program crashes during execution

Understanding Runtime Errors

- Occur when the program crashes during execution
- Caused by **bugs, unexpected input**, or **hardware issues**
- Check the exception message and the line number to troubleshoot

Example of A Runtime Errors

- **ArrayIndexOutOfBoundsException** occurs at runtime because the code attempts to access an index of the array that doesn't exist.



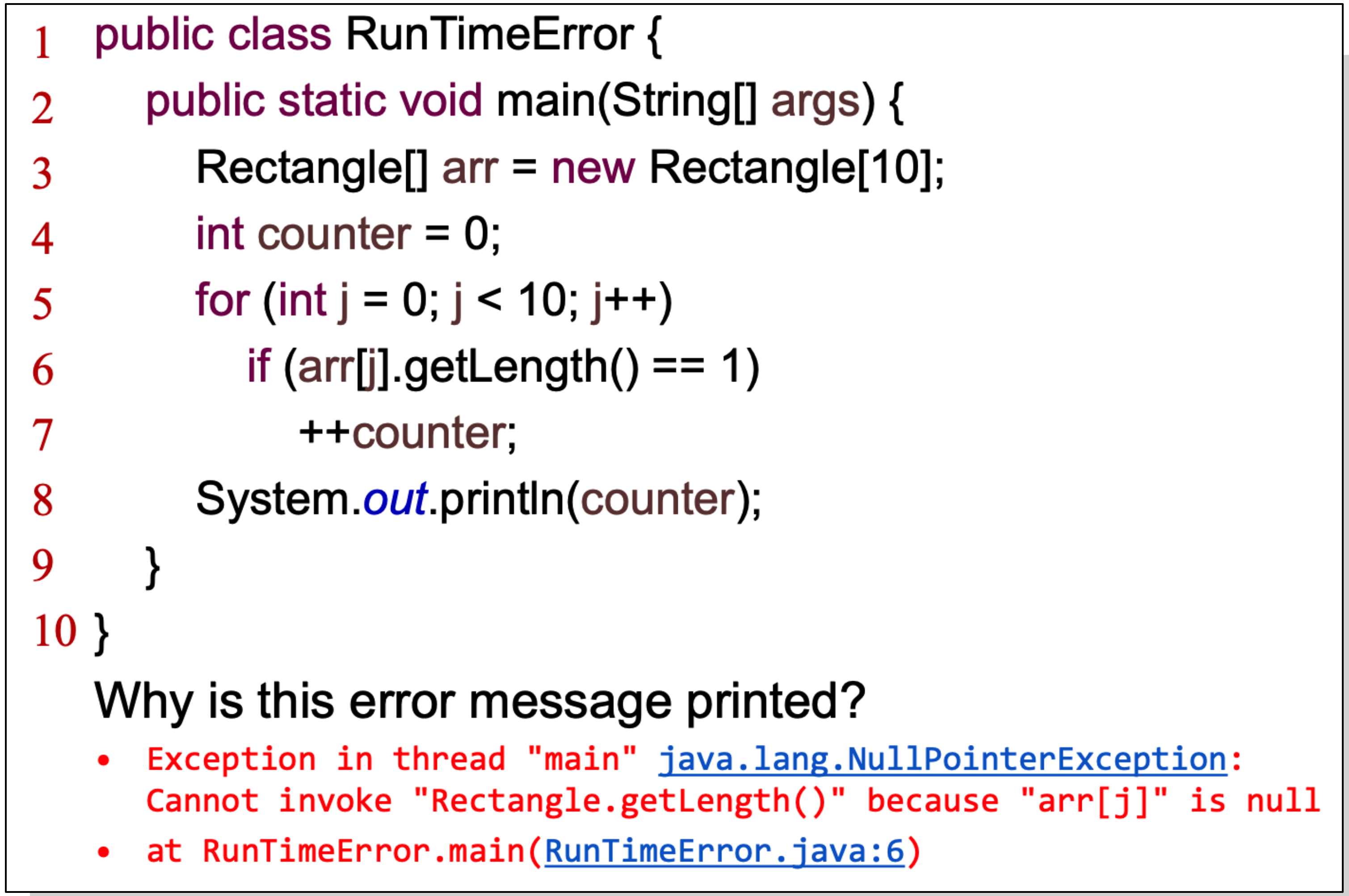
```
1 public class RunTimeError {
2     public static void main(String[] args) {
3         int[] nums = new int[10];
4         for (int j = 0; j <= 10; j++)
5             nums[j] = j;
6     }
7 }
```

This code produces this error message:

```
• Exception in thread "main"
  java.lang.ArrayIndexOutOfBoundsException:
  Index 10 out of bounds for length 10
• at RunTimeError.main(RunTimeError.java:5)
```

Another Example

- **NullPointerException** occurs because you are trying to call a method on a null object.

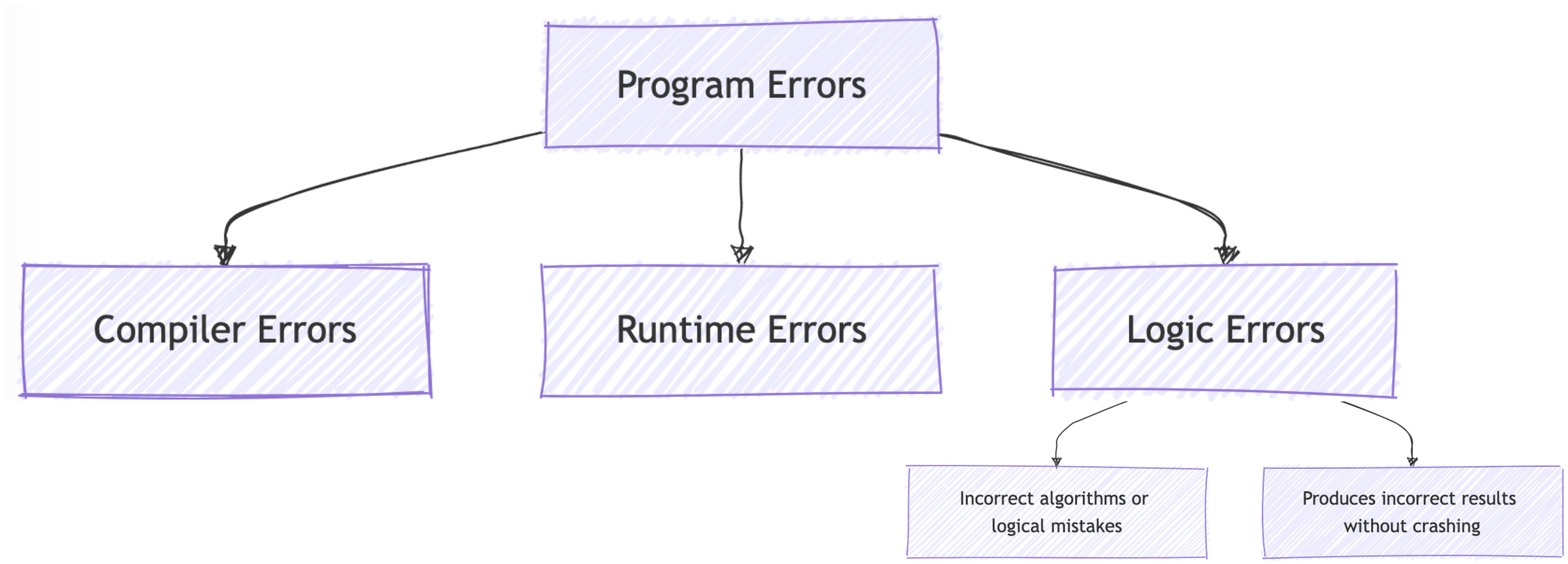


```
1   public class RunTimeError {
2       public static void main(String[] args) {
3           Rectangle[] arr = new Rectangle[10];
4           int counter = 0;
5           for (int j = 0; j < 10; j++)
6               if (arr[j].getLength() == 1)
7                   ++counter;
8           System.out.println(counter);
9       }
10  }
```

Why is this error message printed?

```
• Exception in thread "main" java.lang.NullPointerException:
  Cannot invoke "Rectangle.getLength()" because "arr[j]" is null
• at RunTimeError.main(RunTimeError.java:6)
```

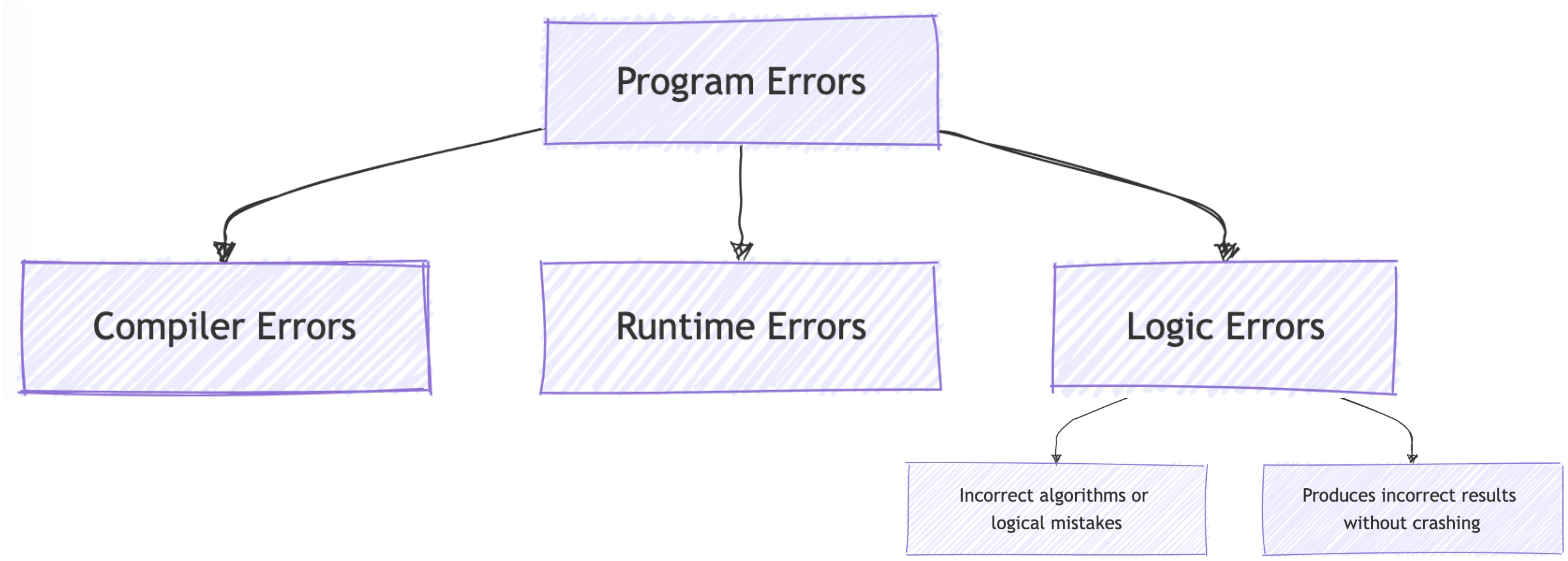
Testing and Debugging (cont.)


Program Errors
Compiler Errors
Runtime Errors
Logic Errors
Incorrect algorithms or logical mistakes
Produces incorrect results without crashing

Identifying and Resolving Runtime Errors

- Check the exception message for the method and line number from which it came.
- Note that the line in the code that caused the exception may not be in line with the error.

Testing and Debugging (cont.)


Program Errors
Compiler Errors
Runtime Errors
Logic Errors
Incorrect algorithms or logical mistakes
Produces incorrect results without crashing

Common Logic Errors

- Using == instead of a method like equals to compare the content of objects
- infinite loops
- Misunderstanding precedence of mathematical operators
- Starting or ending at the wrong index of an array
- Misplaced parentheses (so code is either inside a block when it should not be or vice versa)

Common Logic Errors: Declare Multiple Variables with the Same Name

- Try not to declare multiple variables with the same **name**, as this might lead to program errors.
- Example:

```
// instance variable
private int numStudents;
…
public void someMethod(){
// not the instance variable!
int numStudents = …;
…
}
```

Common Logic Errors: Infinite Loops 1/2

- Another kind of runtime error that can occur is an infinite loop. The program doesn't crash but your program runs infinitely until you manually stop it.
- Example:

```
int i = 0;
while (i < 100) {
    int x = (i + 10) * 25;
}
```

- What is the problem here?
  - There is no update to the variable i inside the loop.

Common Logic Errors: Infinite Loops 2/2

```
boolean done = false;
int a = 25000, b = 0;
while (!done) {
  a = a / 10;
  if (a < b) done = true;
}
```

- What is the problem here?

  The condition a < b will never be true

  (because a `==` b).

Infinite Loops

- Detecting and debugging infinite loops can be challenging. They frequently occur in while loops where the condition remains true indefinitely.
- A useful strategy for identifying these loops is to add a counter that forces the loop to stop after a certain number of iterations.

```
boolean done = false;
int a = 25000, b = 0;
// probably big enough
int maxCheck = 10000;
while (!done && maxCheck > 0) {
   a = a / 10;
   if (a < b) done = true;
      maxCheck--;
}
if (maxCheck == 0)
  System.out.println("probably infinite
loop");
```

Testing & Debugging

- **Testing**: to identify any problems before software is put to use
  - “Testing can show the presence of bugs but can never show their absence”.
- **Debugging**: locating bugs and fixing them

Hints for Success

When writing code:

- Make sure your algorithm is correct before you start coding.

Start small:

- Write and test first simpler methods (e.g. getters, setters, toString). Then, write and test each of the more complex methods individually
- Check your code first with a preliminary hand trace
- Then try running it

Debugging Strategies

- Trace or run your code by hand
- When testing, add a **main** method to each class and invoke all other methods from the **main** method to check that they work as expected. Once you are done testing, delete these main methods.
- Add **print** statements to your code
- Use a debugger (we have already reviewed that!)

Defensive Programming

Write robust programs:

- Include checking for **exceptional conditions**;
- try to think of situations that might reasonably happen, and check for them
  - Examples: files that don't exist, bad input data

Generate appropriate error messages and allow the user to re-enter the data or exit the program.

Test-Driven Development (TDD)

Tests are written before the actual code!

Test-based Programming in Java

- Test-based programming in Java, often called Test-Driven Development (TDD), is a development approach where tests are written before the actual code is implemented.
- Java provides several tools and frameworks for test-based programming, with **JUnit** being one of the most popular.
- **JUnit** allows developers to create unit tests that can automatically verify the correctness of small code components, like methods or classes

The process typically involves the following steps:

1. **Write a Test**: Create a unit test to define the desired functionality.
2. **Write the Code**: Write just enough code to pass the test.
3. **Run the Test**: Ensure the code works as expected.
4. **Refactor**: Improve the code while keeping the test passing.
5. **Repeat**: Continue the process with new tests for additional features.

Test-based Programming in Java

```java
import static org.junit.Assert.assertEquals;
import org.junit.Test;

public class CalculatorTest {

    @Test
    public void testMultiply() {
        Calculator calculator = new Calculator();
        int result = calculator.multiply(3, 4);
        // Expect 3 * 4 to equal 12
        assertEquals(12, result);
    }
}
-------------------------------------------------------
public class Calculator {
    public int multiply(int a, int b) {
        return a * b;
    }
}
```

The process typically involves the following steps:

1. **Write a Test**: Create a unit test to define the desired functionality.
2. **Write the Code**: Write just enough code to pass the test.
3. **Run the Test**: Ensure the code works as expected.
4. **Refactor**: Improve the code while keeping the test passing.
5. **Repeat**: Continue the process with new tests for additional features.

Thank you