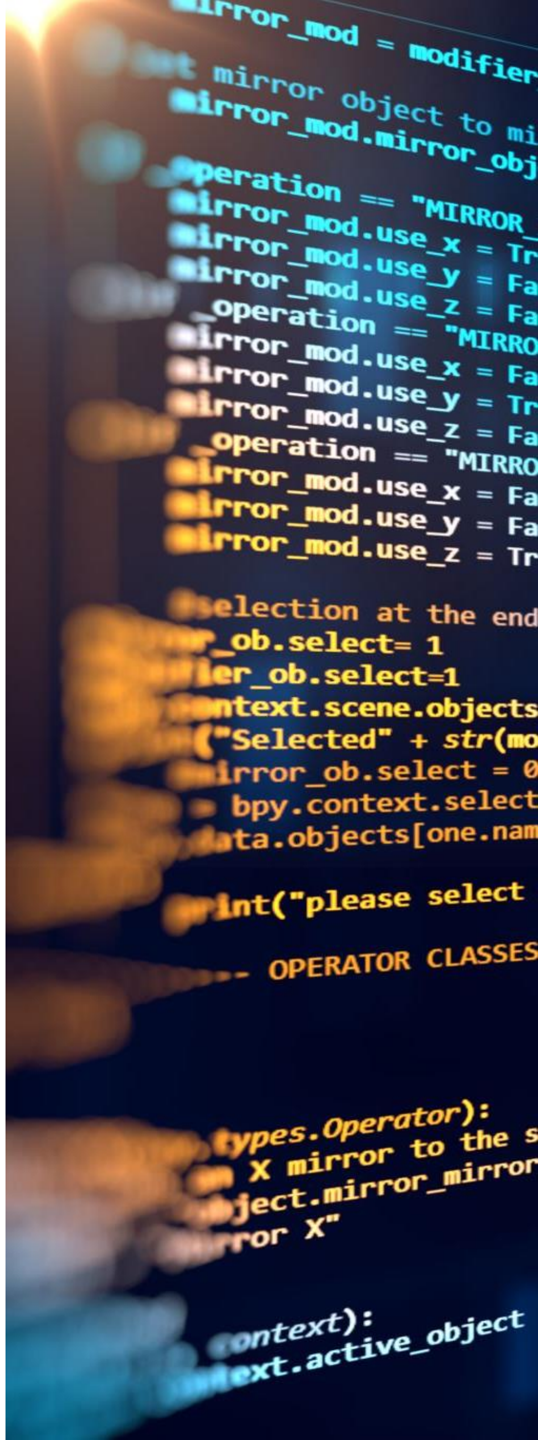
Please use the following QR code to check in and record your attendance.

CS 1027
Fundamentals of Computer Science II

Inheritance in Java (cont.)

Ahmed Ibrahim



Recap: Inheritance in Java

Inheritance Basics

- **Subclass** – A new class created from an existing class (Superclass), **inheriting** its properties and methods.
- **Superclass** – The parent/base class from which the subclass derives.

Key Benefits

- **Reusability**: Reuse code from existing classes.
- **Maintainability**: Easier to manage and update code.
- **Flexibility** & **Encapsulation**: Modify and extend existing classes.

Core Concepts Covered

Is-a Relationship

- A subclass is a specialized version of its superclass.
- Example: A Square is a Rectangle, but a Rectangle isn't always a Square.

Polymorphism

- Allows a variable of the superclass type to reference an object of the subclass type.
- Enables writing general, flexible code that works with different subclasses.

Question!

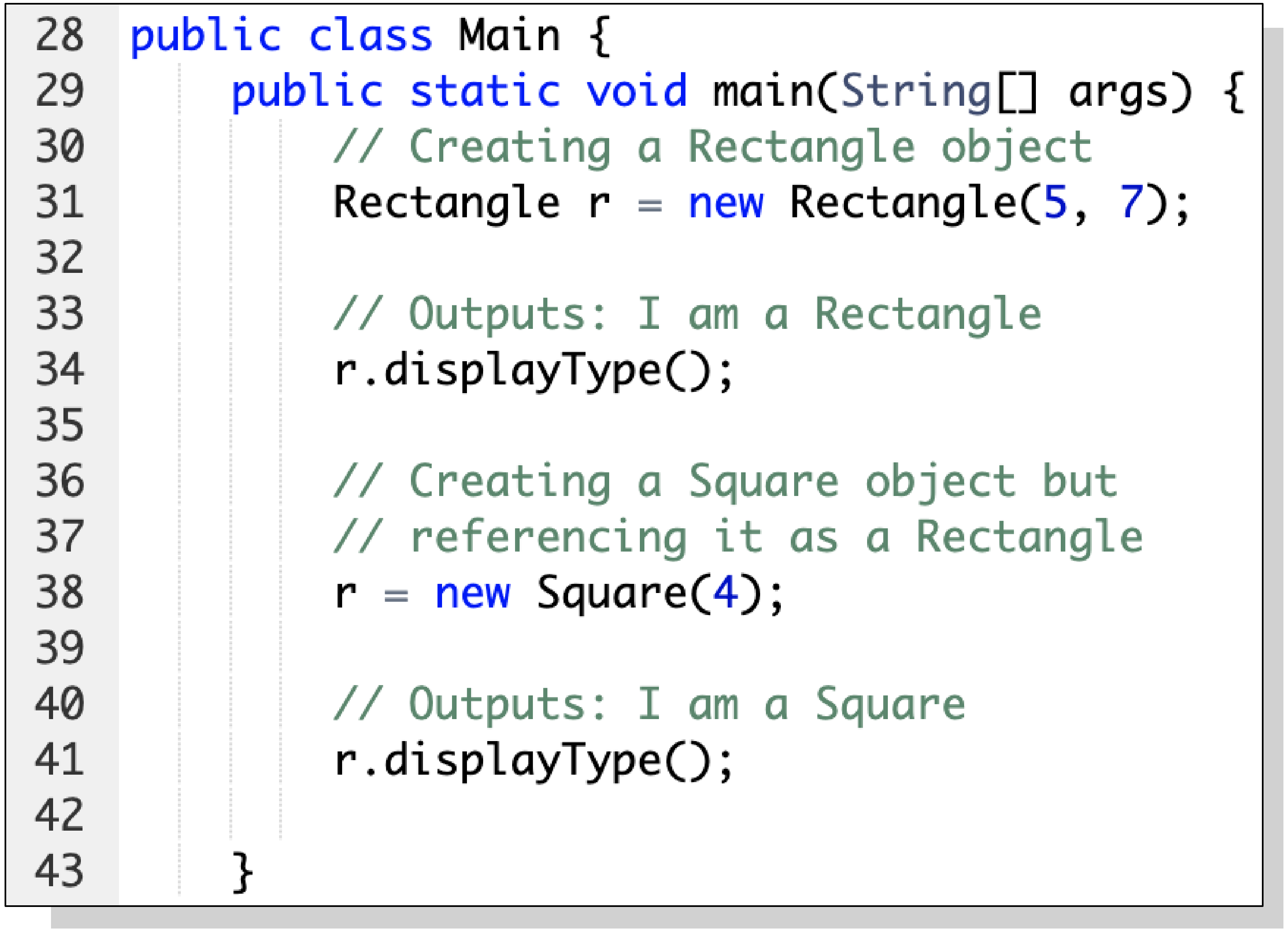
- Imagine you have a superclass called Shape with a method draw(). You also have three subclasses: Circle, Rectangle, and Triangle, each with its own version of the draw() method.
- Code snippet:
- Which version of the draw() method would be called for each object?

```
Shape shape1 = new Circle();
Shape shape2 = new Rectangle();
Shape shape3 = new Triangle();

shape1.draw();
shape2.draw();
shape3.draw();
```

Dynamic Binding

- **Dynamic binding**, also known as late binding, is a mechanism in which the method to be invoked is determined at runtime based on the actual object's type rather than the reference type.
- Without dynamic binding, the program would always call the draw() method from the Shape superclass, regardless of the actual object's type.
- Dynamic binding solves this problem by allowing the program to decide at runtime which version of the draw() method to call, based on the actual object type (Circle, Rectangle, or Triangle).
- This ensures the correct, **overridden** method is executed, allowing polymorphism to work as intended.

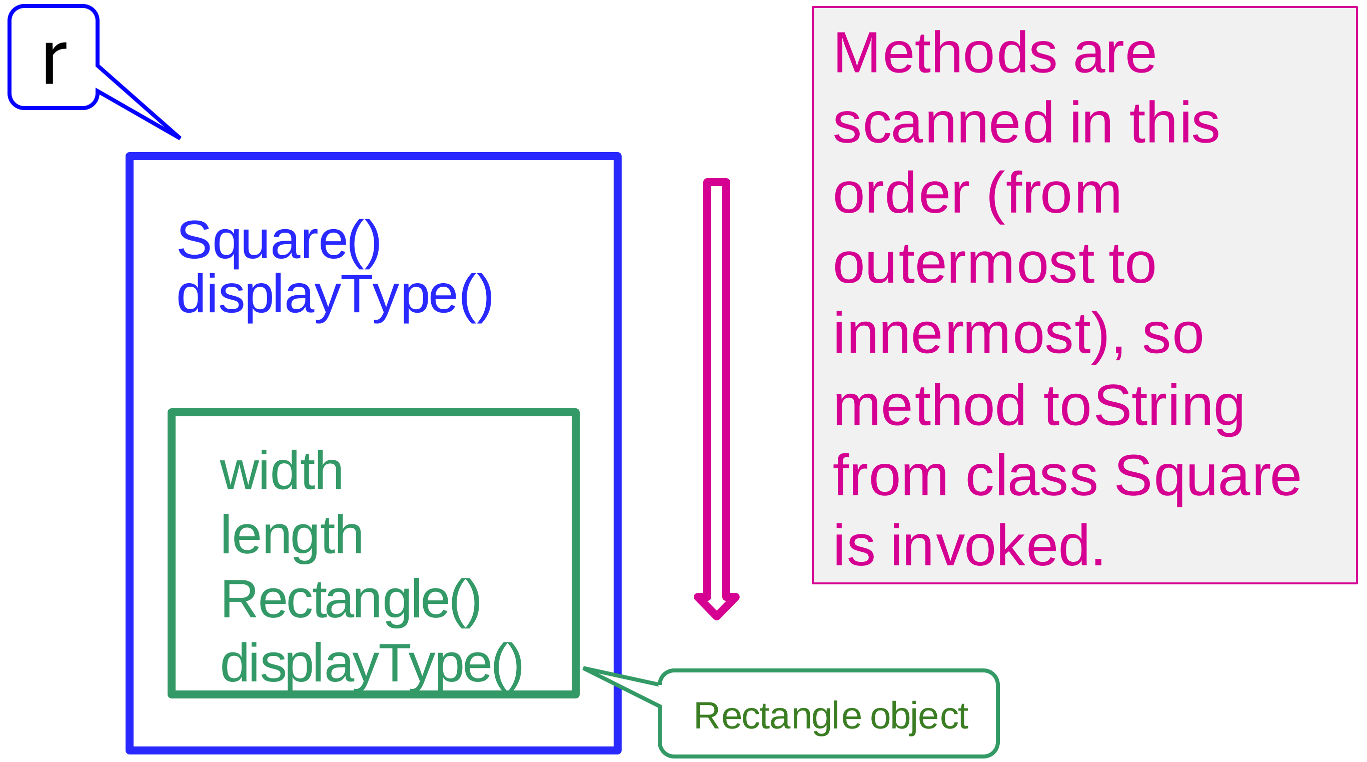
Another Example



```java
public class Main {
    public static void main(String[] args) {
        // Creating a Rectangle object
        Rectangle r = new Rectangle(5, 7);

        // Outputs: I am a Rectangle
        r.displayType();

        // Creating a Square object but
        // referencing it as a Rectangle
        r = new Square(4);

        // Outputs: I am a Square
        r.displayType();

    }
```

Dynamic Binding

- When a method is present in both the superclass and the subclass, the version from the subclass is executed.
- The method that is called must be defined in the superclass (or one of its parent classes); otherwise, a **compiler error** will occur.


r
Square()
displayType()
width
length
Rectangle()
displayType()
Rectangle object
Methods are scanned in this order (from outermost to innermost), so method toString from class Square is invoked.

Type Casting

Review: Casting Primitive Types

- We can use casting to convert some primitive types to other primitive types.
- Example:

```
int i, j, n;
n = (int) Math.random( );
double q = (double) i / (double) j;
```

- Note that this actually changes the representation from double to integer (second statement) or from integer to double (last statement).

Casting Reference Variables: **Upcasting**

- Recall:

```
//Here, r is created as a Square object
Rectangle r = new Square(5);
```

- Upcasting: Assigning a subclass object to a superclass reference (child to a parent).
- This is done implicitly.

Casting Reference Variables: **Downcasting**

- **Downcasting**: Converting a superclass reference back to a subclass reference (parent to a child).
- This must be done explicitly.
- Example:

```
//Here, r is created as a Square object
Rectangle r = new Square(5);
Square s = (Square) r; // downcasting
```

- The compiler checks in the background to see if this type of casting is possible or not.
- If it's not possible, the compiler throws a ClassCastException.

The Need for Downcasting

- Go back to the example:

```
Rectangle r = new Square(5);
System.out.println(r.getSide());
```

- This will generate a compiler error (why?)
  - The compiler error occurs because r is declared a Rectangle and lacks a **getSide**() method.
  - Although **r** references a Square object, the compiler only recognizes **r** as a Rectangle and restricts access to Square-specific methods.
- So, how do we overcome this? The answer lies in downcasting.
  - `System.out.println(((Square) r).getSide());`
- We can let the compiler know that we intend variable **r** to reference a Square object, by casting it to type Square.
- Remember: Casting does not change the object being referenced!

Question!

Why might the following code result in a runtime error?

```
Animal a = new Animal();
Dog d = (Dog) a;
```

A) Because **a** is not an instance of **Dog**

B) Because **a** must be declared as a **Dog** for the cast to work

C) Because downcasting is only valid within the same class

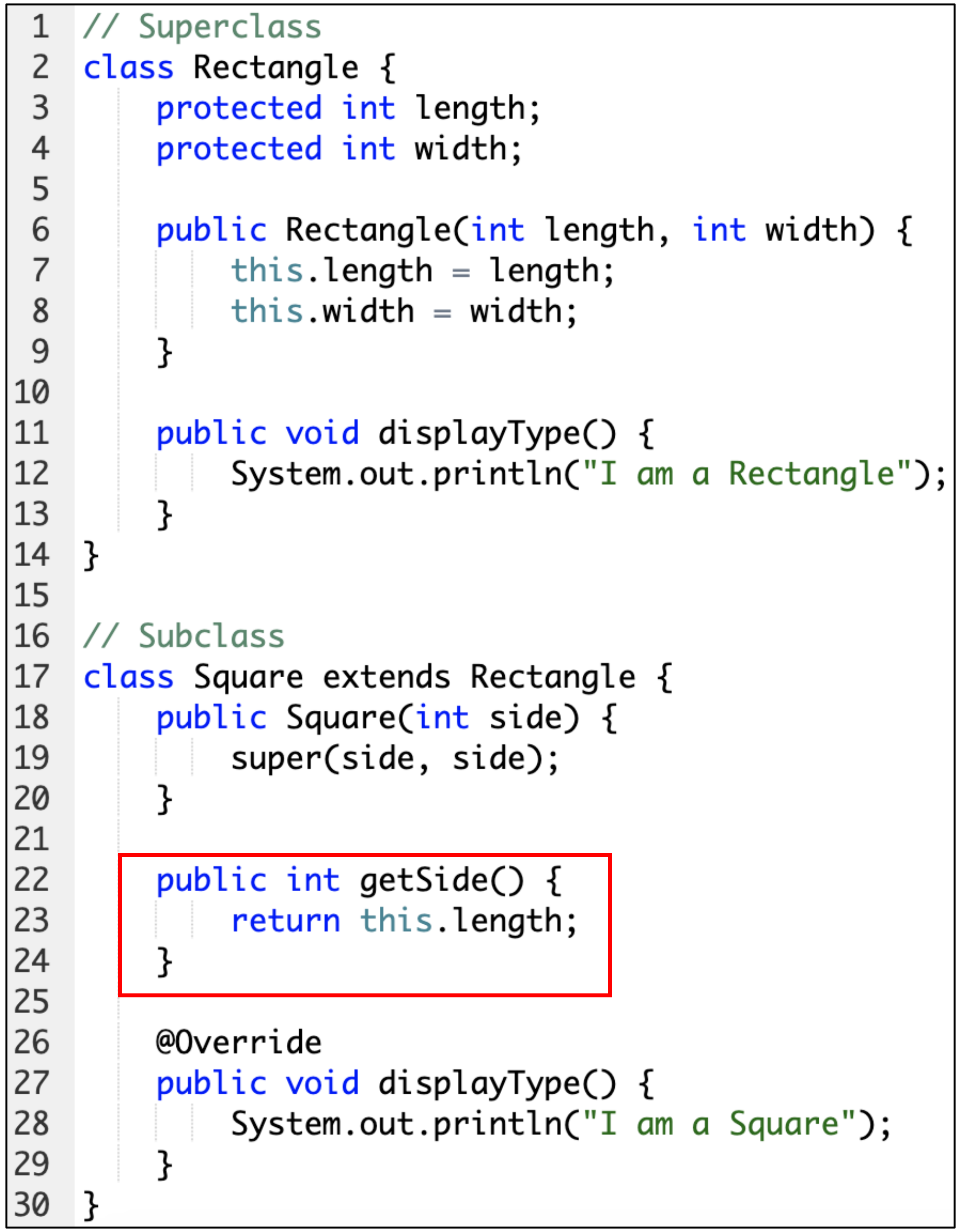
D) Because **a** and **d** must be of the same type

This can only be done if the object type being referenced is actually an instance of the subclass, like in slide 12

Recall:

Square Class

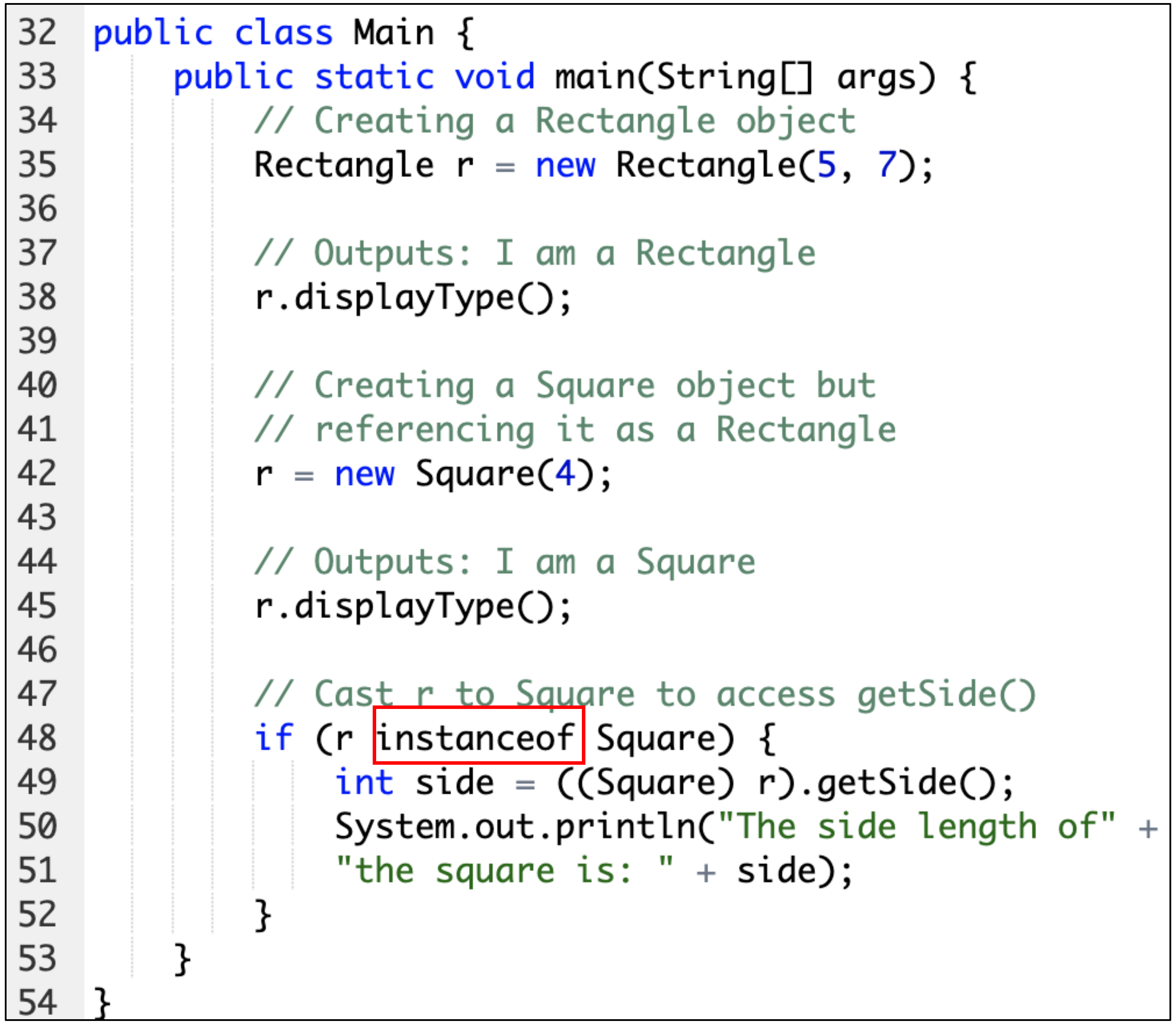
With modification



```java
// Superclass
class Rectangle {
    protected int length;
    protected int width;

    public Rectangle(int length, int width) {
        this.length = length;
        this.width = width;
    }

    public void displayType() {
        System.out.println("I am a Rectangle");
    }
}

// Subclass
class Square extends Rectangle {
    public Square(int side) {
        super(side, side);
    }

    public int getSide() {
        return this.length;
    }

    @Override
    public void displayType() {
        System.out.println("I am a Square");
    }
}
```

instanceof Operator

- What if **r** was not actually referencing a Square object at the time of casting?
- The compiler would accept it, but a **runtime error** would occur.



```
public class Main {
    public static void main(String[] args) {
        // Creating a Rectangle object
        Rectangle r = new Rectangle(5, 7);

        // Outputs: I am a Rectangle
        r.displayType();

        // Creating a Square object but
        // referencing it as a Rectangle
        r = new Square(4);

        // Outputs: I am a Square
        r.displayType();

        // Cast r to Square to access getSide()
        if (r instanceof Square) {
            int side = ((Square) r).getSide();
            System.out.println("The side length of" +
            "the square is: " + side);
        }
    }
}
```

instanceof Operator (cont.)

- A safer fix: use the instanceof operator

```
if (r instanceof Square) {
   System.out.println(((Square)r).getSide( ));
}
```

- Note that instanceof is an operator, not a method
  - An operator is a built-in language feature used to perform a specific operation. In this case, instanceof checks whether an object is an instance of a particular class or interface.
- It tests whether the referenced object is an instance of a particular class and gives the expression the value TRUE or FALSE.

Exercise!

Given the following three Java classes:

```
public class Animal
public class Dog extends Animal
public class Cat extends Animal
```

And the following variables:

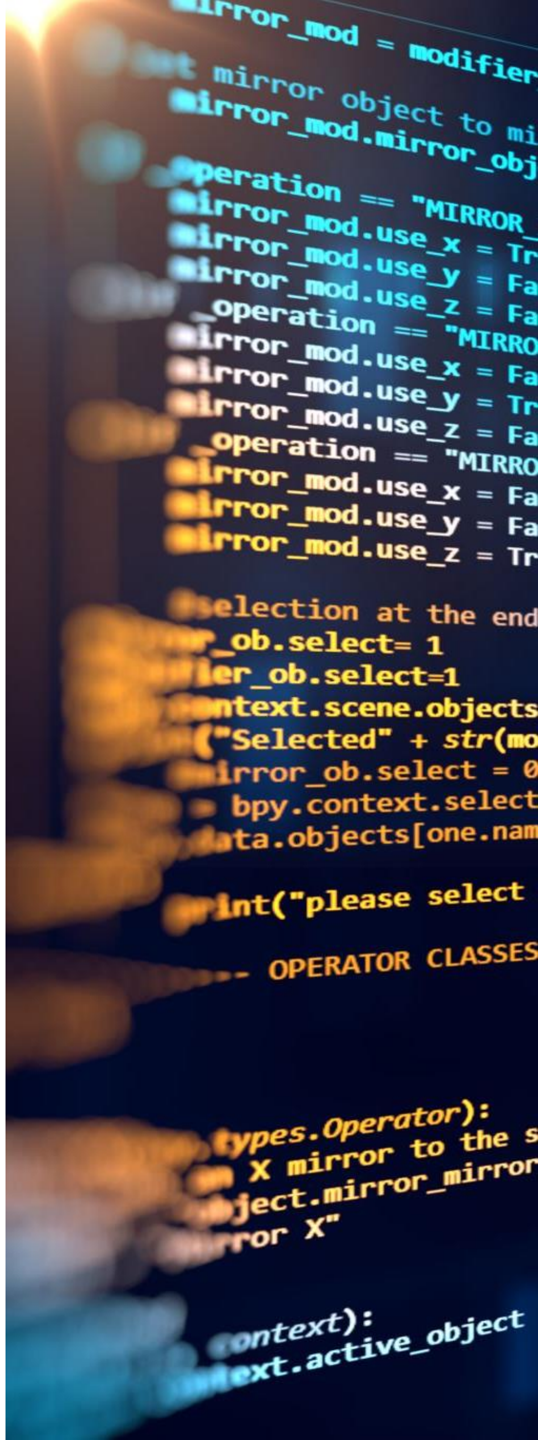
```
Animal animalVar;
Dog dogVar;
Cat catVar;
```

Determine which of the following statements are correct, which generate compilation errors, and which cause runtime errors:

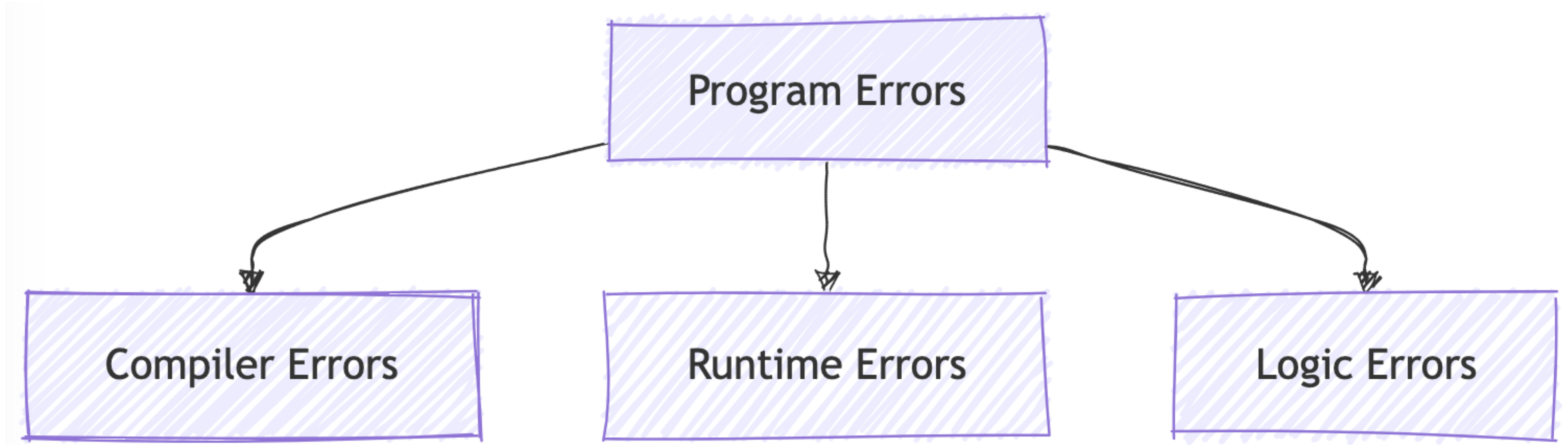
```
1. animalVar = new Dog();
2. animalVar = new Cat();
3. dogVar = new Cat();
4. dogVar = new Animal();
5. catVar = new Dog();
```

Debugging

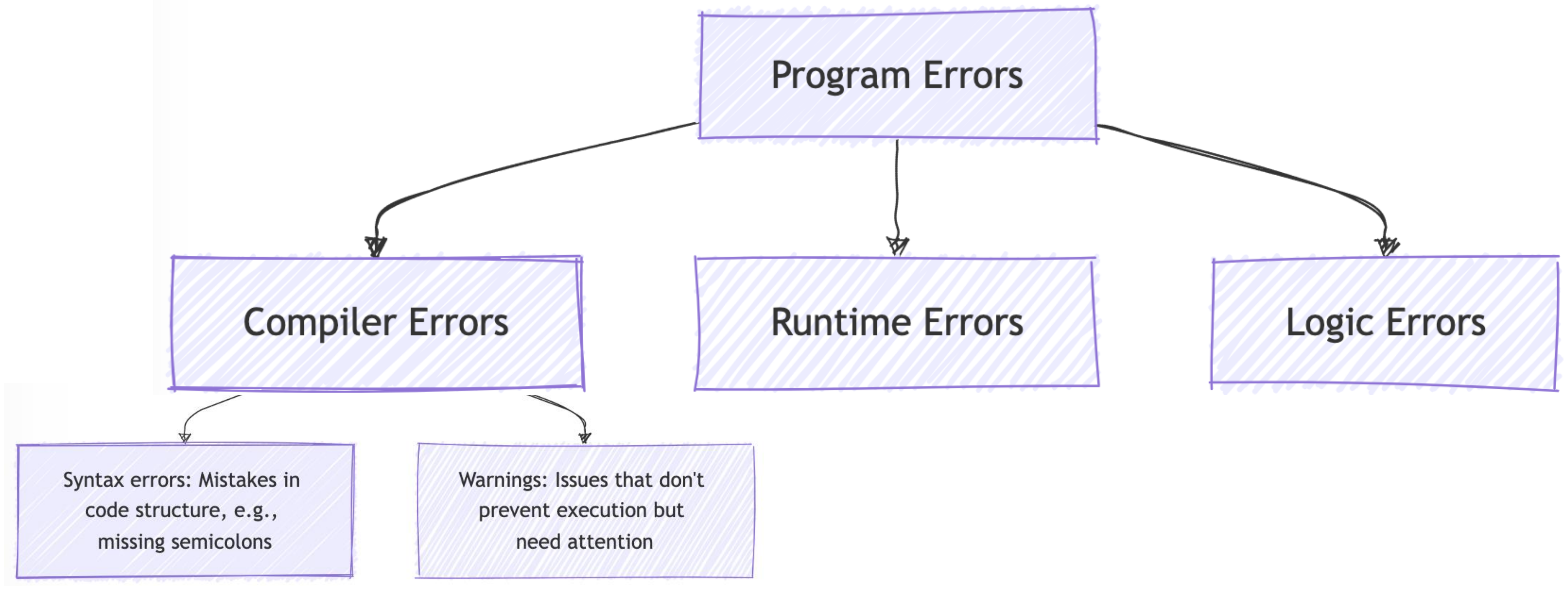
Understanding and Fixing Program Errors



Testing and Debugging



Testing and Debugging

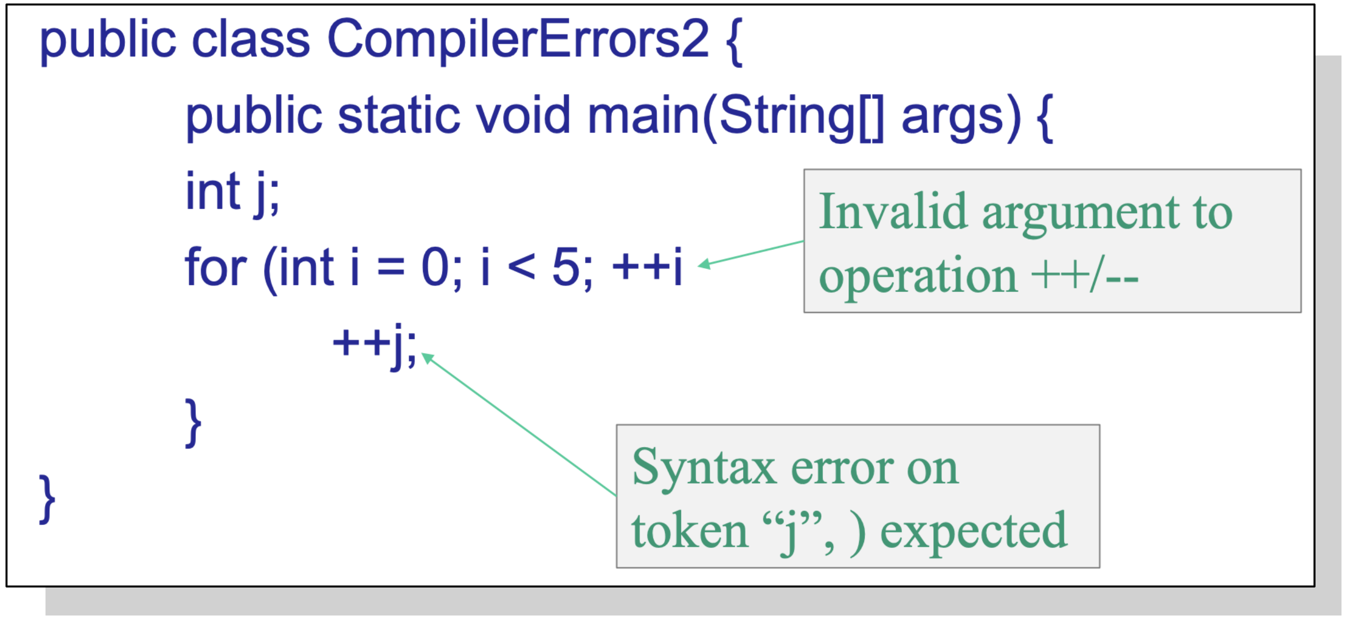

Program Errors
Compiler Errors
Runtime Errors
Logic Errors
Syntax errors: Mistakes in code structure, e.g., missing semicolons
Warnings: Issues that don't prevent execution but need attention

Understanding Compiler Errors

- Errors are detected by the compiler when the code does not follow the correct syntax or language rules.

Syntax Errors	Type Mismatch	Undefined Variables or Methods	Redeclaration Errors
Mistakes in the code structure, such as missing semicolons, brackets, or incorrect variable declarations.	Using variables or methods in ways that don't match their declared data types.	Trying to use variables or methods that haven’t been declared or imported.	Declaring a variable that has already been declared.

Can you explain these two error messages?

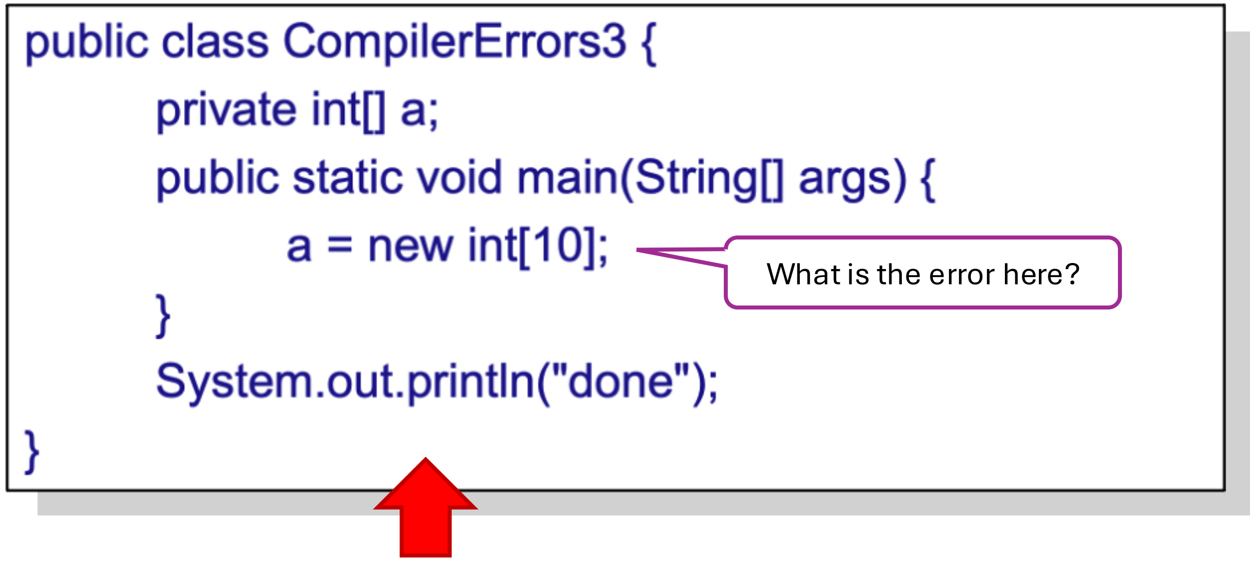


```
public class CompilerErrors2 {
        public static void main(String[] args) {
        int j;
        for (int i = 0; i < 5; ++i
                ++j;
        }
}
```

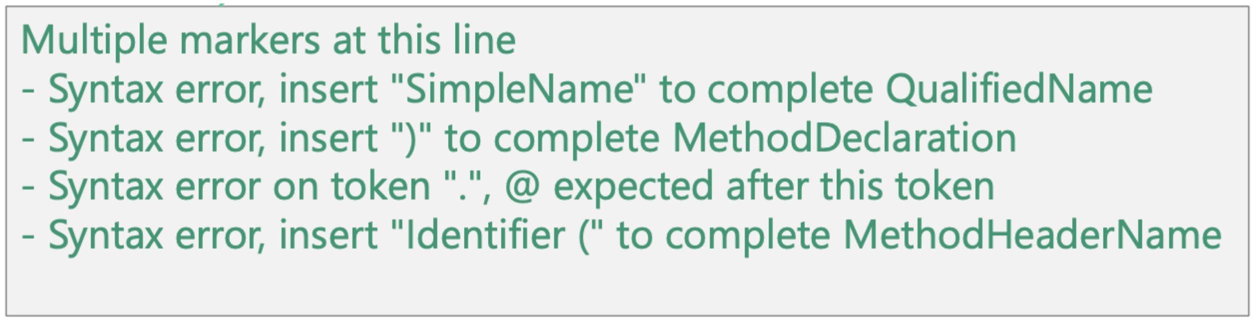
Invalid argument to operation ++/--

Syntax error on token “j”,) expected

Another Example



```
public class CompilerErrors3 {
        private int[] a;
        public static void main(String[] args) {
                a = new int[10];
        }
        System.out.println("done");
}
```



```
Multiple markers at this line
- Syntax error, insert "SimpleName" to complete QualifiedName
- Syntax error, insert ")" to complete MethodDeclaration
- Syntax error on token ".", @ expected after this token
- Syntax error, insert "Identifier (" to complete MethodHeaderName
```

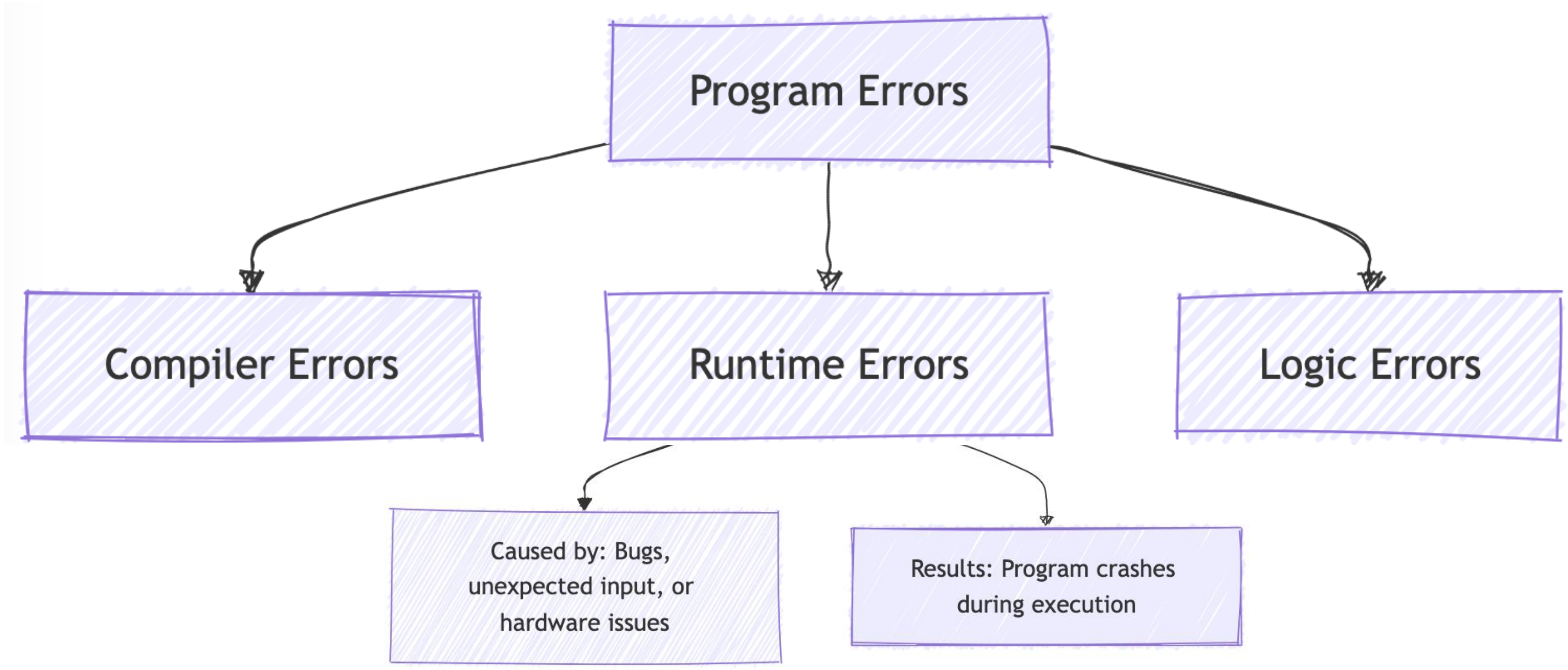
Why Are Compiler Errors Confusing?

- **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.

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.)

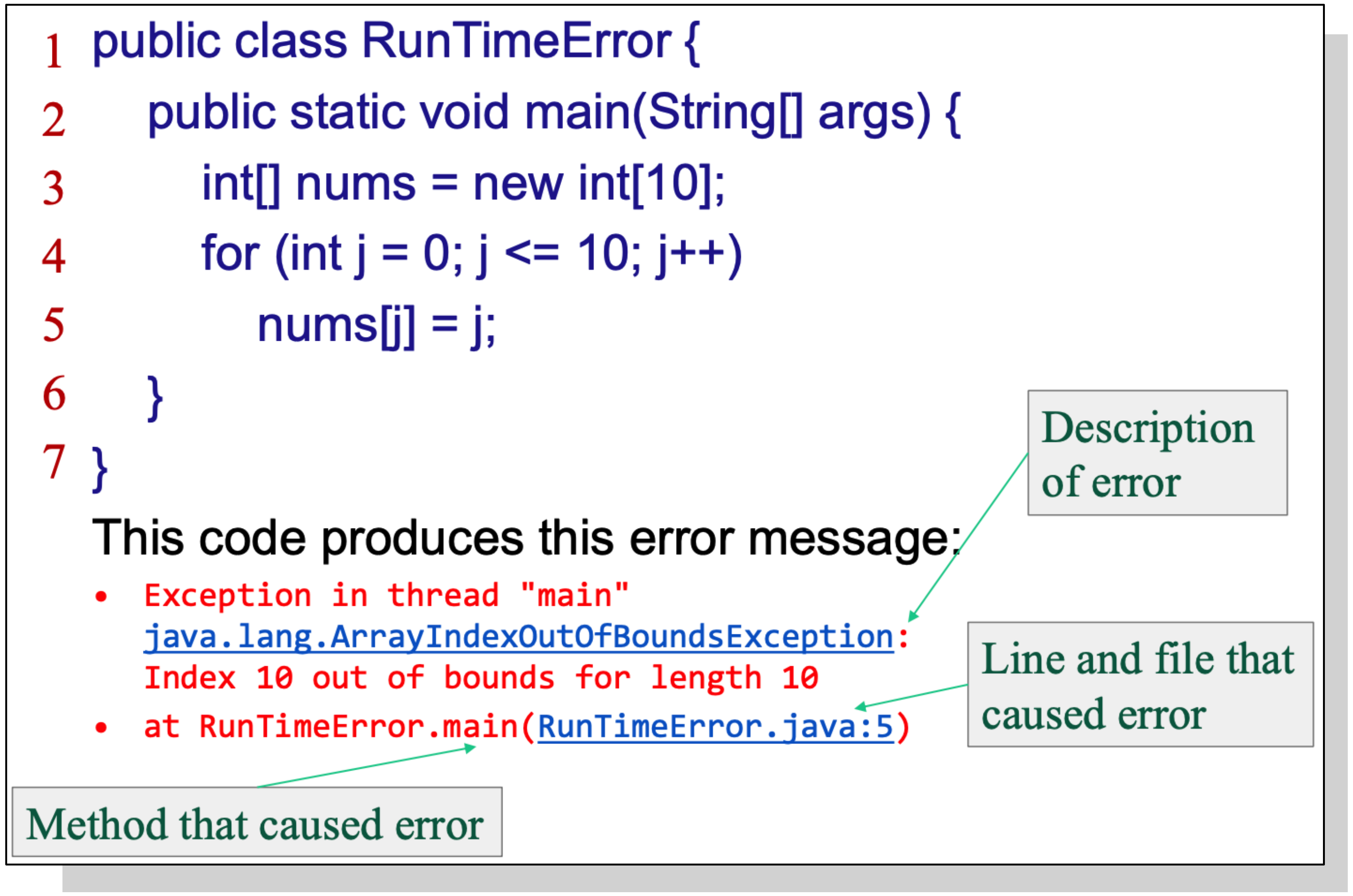

Program Errors
Compiler Errors
Runtime Errors
Logic Errors
Caused by: Bugs, unexpected input, or hardware issues
Results: Program crashes during execution

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.



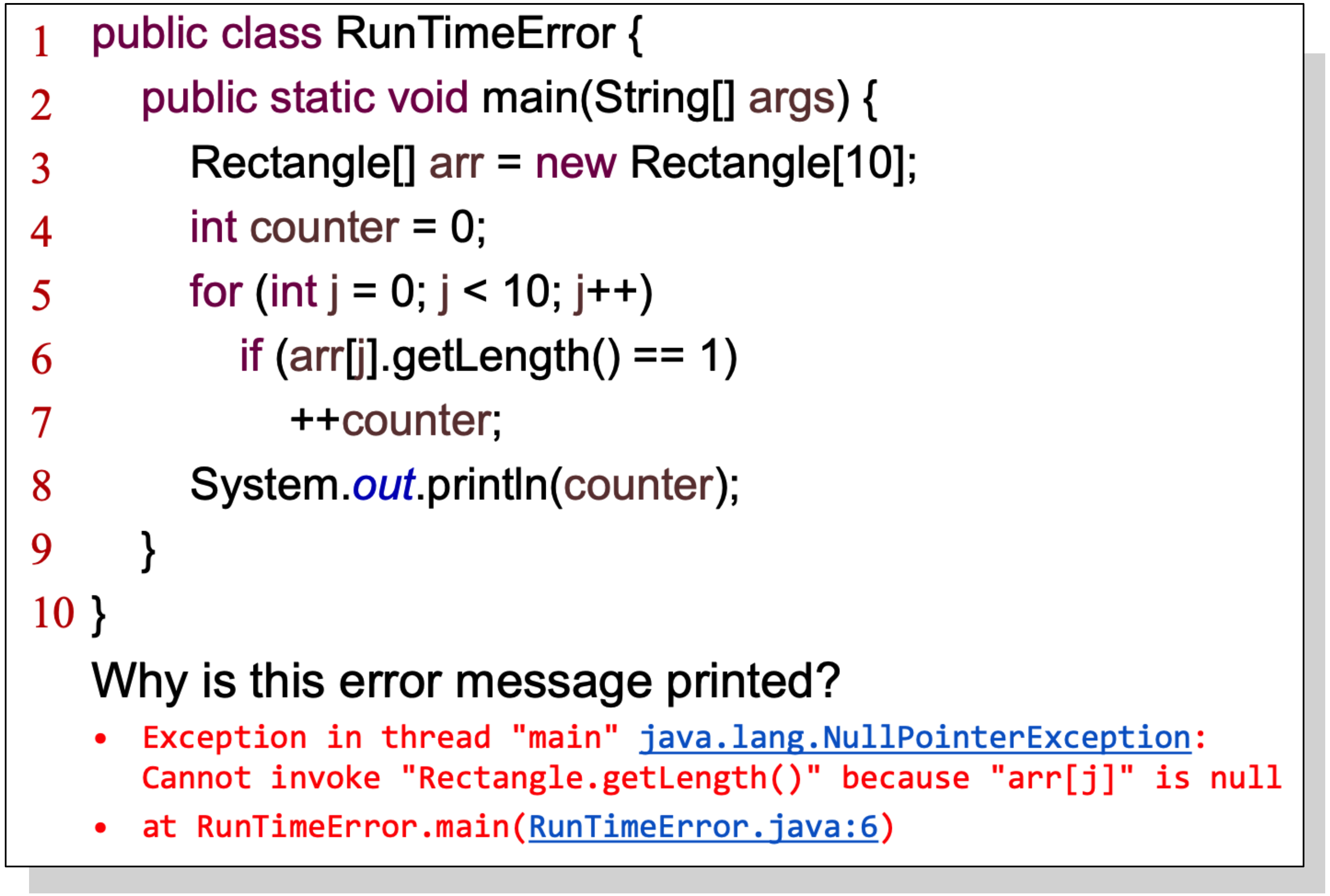
```
1 public class RunTimeError {
2     public static void main(String[] args) {
3         int[] nums = new int[10];
4         for (int j = 0; j <= 10; j++)
5             nums[j] = j;
6     }
7 }
```

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.

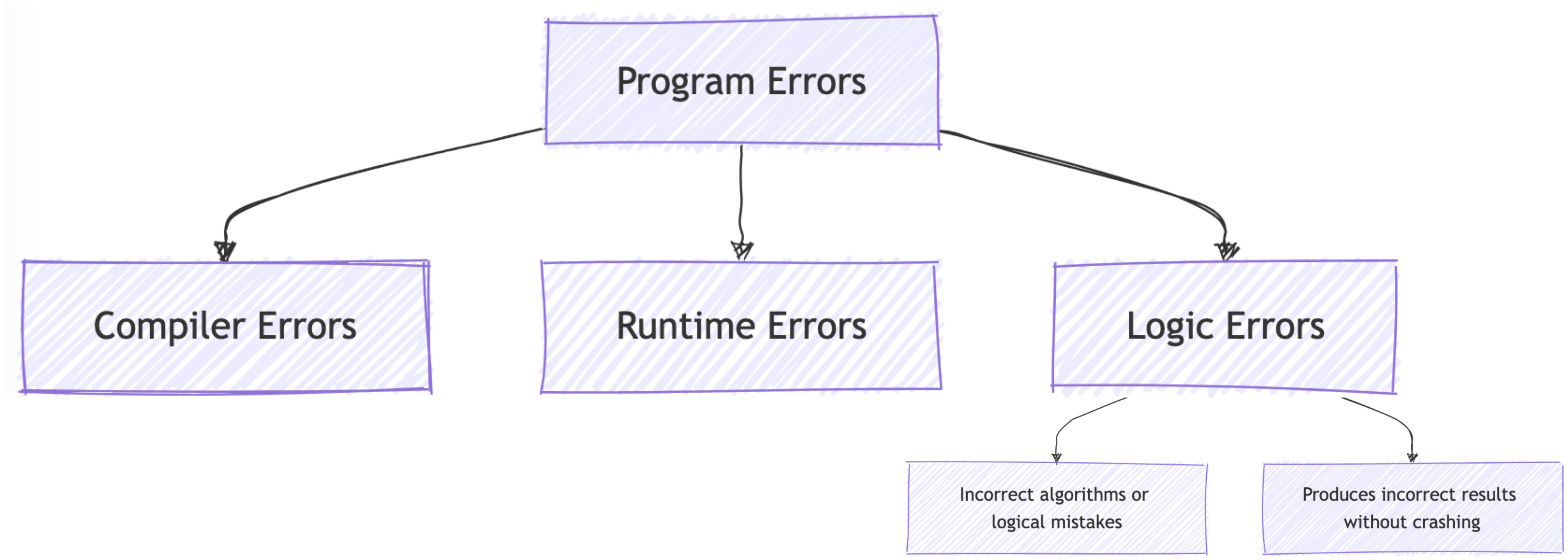


```
1   public class RunTimeError {
2       public static void main(String[] args) {
3           Rectangle[] arr = new Rectangle[10];
4           int counter = 0;
5           for (int j = 0; j < 10; j++)
6               if (arr[j].getLength() == 1)
7                   ++counter;
8           System.out.println(counter);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

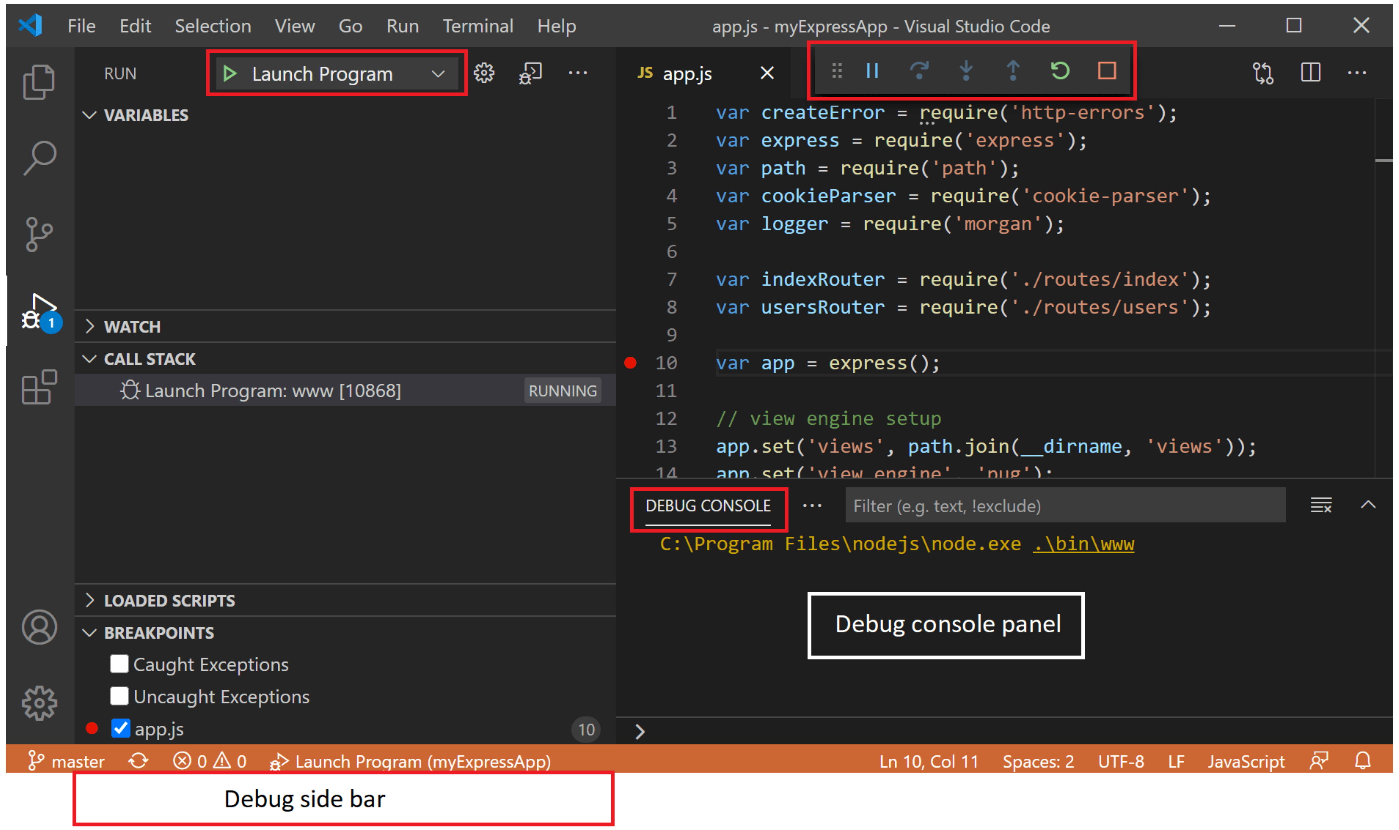
Testing and Debugging (cont.)


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

Using the IDE Debugger

Start debugging

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


File Edit Selection View Go Run Terminal Help
app.js - myExpressApp - Visual Studio Code
RUN
Launch Program
app.js
VARIABLES
WATCH
CALL STACK
Launch Program: www [10868] RUNNING
LOADED SCRIPTS
BREAKPOINTS
Caught Exceptions
Uncaught Exceptions
app.js 10
DEBUG CONSOLE
Filter (e.g. text, !exclude)
master
0 0
Launch Program (myExpressApp)
Ln 10, Col 11 Spaces: 2 UTF-8 LF JavaScript
Debug console panel
Debug side bar

```
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
```

Thank you