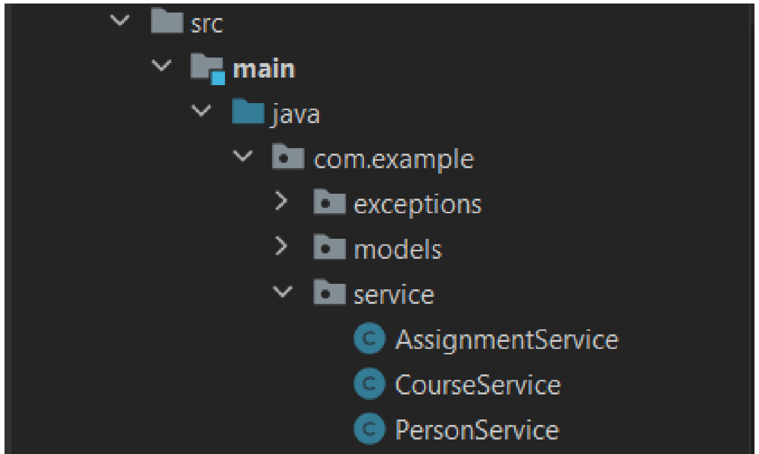
Week 1 Day 3

Java Fundamentals

REVATURE

Java: Package and Imports

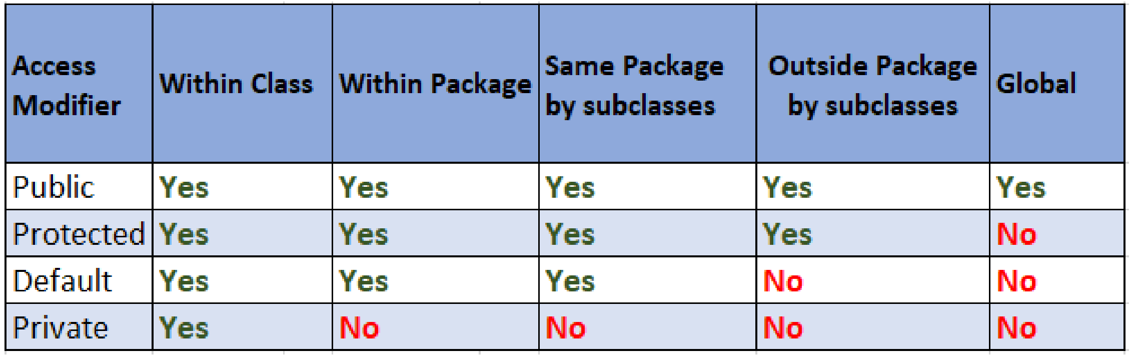
- Packages create a folder structure for your project
  - Use lowercase reverse domain naming
    - com.example.package
- Import packages with import statements inside of classes
  - java.lang is automatically imported to all classes



```
src
  main
    java
      com.example
        exceptions
        models
        service
          AssignmentService
          CourseService
          PersonService
```

Java: Access Modifiers

- Four access modifiers which restrict access to different layers of the application
- They can be placed in front of
  - Classes
  - Interfaces
  - Enums
  - Class members



Access Modifier	Within Class	Within Package	Same Package by subclasses	Outside Package by subclasses	Global
Public	Yes	Yes	Yes	Yes	Yes
Protected	Yes	Yes	Yes	Yes	No
Default	Yes	Yes	Yes	No	No
Private	Yes	No	No	No	No

Image Source: https://javahungry.blogspot.com/2019/12/access-modifiers-in-java.html

Java: Non-Access Modifiers

- Gives java extra functionality, important ones include
  - Static: gives class member class/static scope
  - Final: denotes the member is unchangeable
  - Abstract: classes can not longer be directly instantiated, methods cannot be implemented
  - Transient: marks class member as non-serializable

Java: Variable Scopes

- Instance Scope
  - Belongs to the object
- Class/Static Scope
  - Belongs to the class
- Method Scope
  - Only available in the method block
- Block Scope
  - Only available in the specific block of code

Java: Object Class

Object class is the root of every class in Java

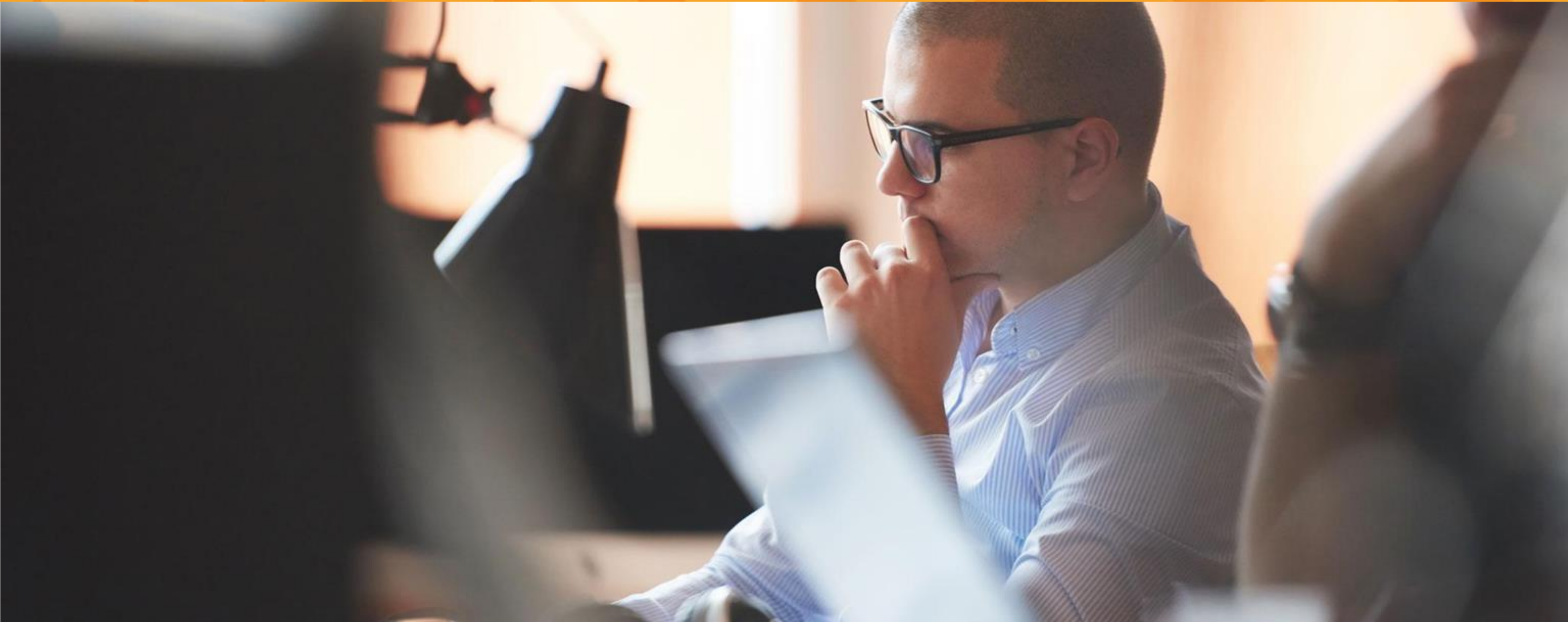
- All java classes will inherently have these methods
  - Object clone()
  - boolean equals(Object o)
  - void finalize()
  - Class<?> getClass()
  - int hashCode()
  - void notify(), notifyAll()
  - String toString()
  - void wait(), wait(long timeout), wait(long timeout, int nanos)

Java: hashCode() and equals()

Commonly overridden for custom class logic

- Object.equals() is overridden to compare properties of the object
  - None overridden .equals compares memory address
- If Object.equals() is overridden Object.hashCode() should also be
  - The result of hashCode() of an object should not change
  - If .equals returns true hash codes are equals
  - If . equals returns false hash codes can be the same OR different

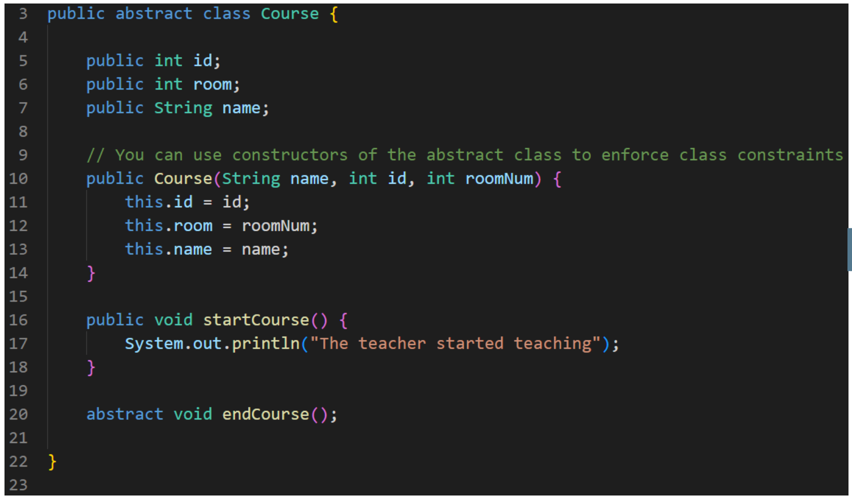
Java Modifiers and Scope Demo



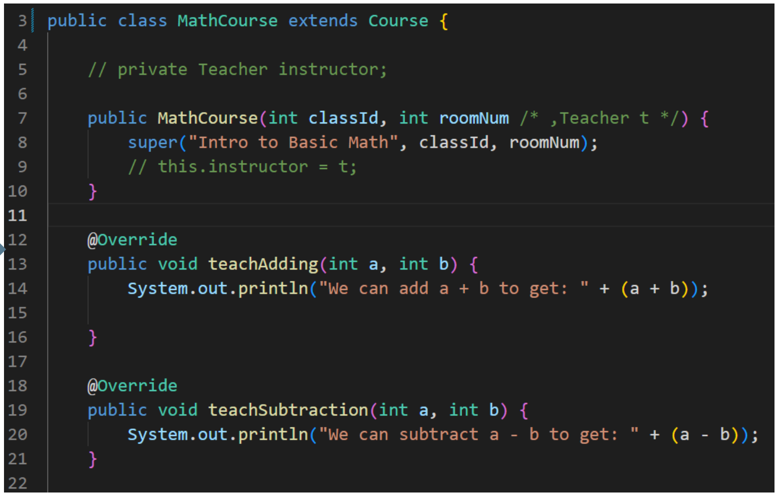
Java: Abstract Classes and Methods

REVATURE

- Abstract Classes
  - General classes that cannot be instantiated
  - Created with the abstract keyword
  - Contains abstract and concrete methods
  - Use the extends keyword



```java
public abstract class Course {

    public int id;
    public int room;
    public String name;

    // You can use constructors of the abstract class to enforce class constraints
    public Course(String name, int id, int roomNum) {
        this.id = id;
        this.room = roomNum;
        this.name = name;
    }

    public void startCourse() {
        System.out.println("The teacher started teaching");
    }

    abstract void endCourse();

}
```



```java
public class MathCourse extends Course {

    // private Teacher instructor;

    public MathCourse(int classId, int roomNum /* ,Teacher t */) {
        super("Intro to Basic Math", classId, roomNum);
        // this.instructor = t;
    }

    @Override
    public void teachAdding(int a, int b) {
        System.out.println("We can add a + b to get: " + (a + b));

    }

    @Override
    public void teachSubtraction(int a, int b) {
        System.out.println("We can subtract a - b to get: " + (a - b));
    }

```

Java: Abstraction via Interfaces

- Interfaces
  - Contracts for classes with methods to implement
  - Inherently public and abstract
  - All methods are public and abstract
  - Variables are public and final
  - Uses the implements Keyword
- Special Interfaces
  - Marker Interfaces
  - Functional Interfaces

REVATURE

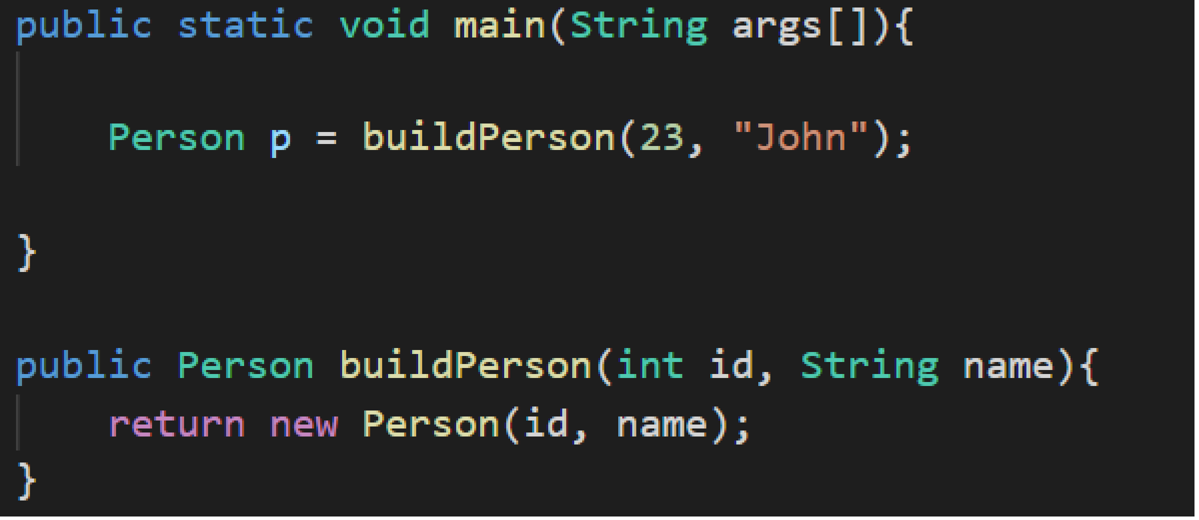
Java Abstract Class and Interface Demo

Java: Casting and Covariance

- Casting allows for objects to act as others
  - Upcasting assigns child object to parent object
  - Downcasting assigns parent object to child object
  - Works because of reference types in memory
- Covariance/Covariant Return Types
  - Allows you to change the child object methods
    - Return type
    - Access modifier
    - Or exception type

Java: Stack vs Heap Memory

- Heap is the total memory for the application
  - Cannot control the amount
  - Contains the stack and objects
  - The new keywords adds new objects to the heap
  - OutOfMemoryError occurs if you run out of heap space
- Stack stores method calls, and local variables
  - Methods and their local variables get put on the top of the stack
  - StackOverflowError occurs if you run out of stack space



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

    Person p = buildPerson(23, "John");

}

public Person buildPerson(int id, String name){
    return new Person(id, name);
}
```

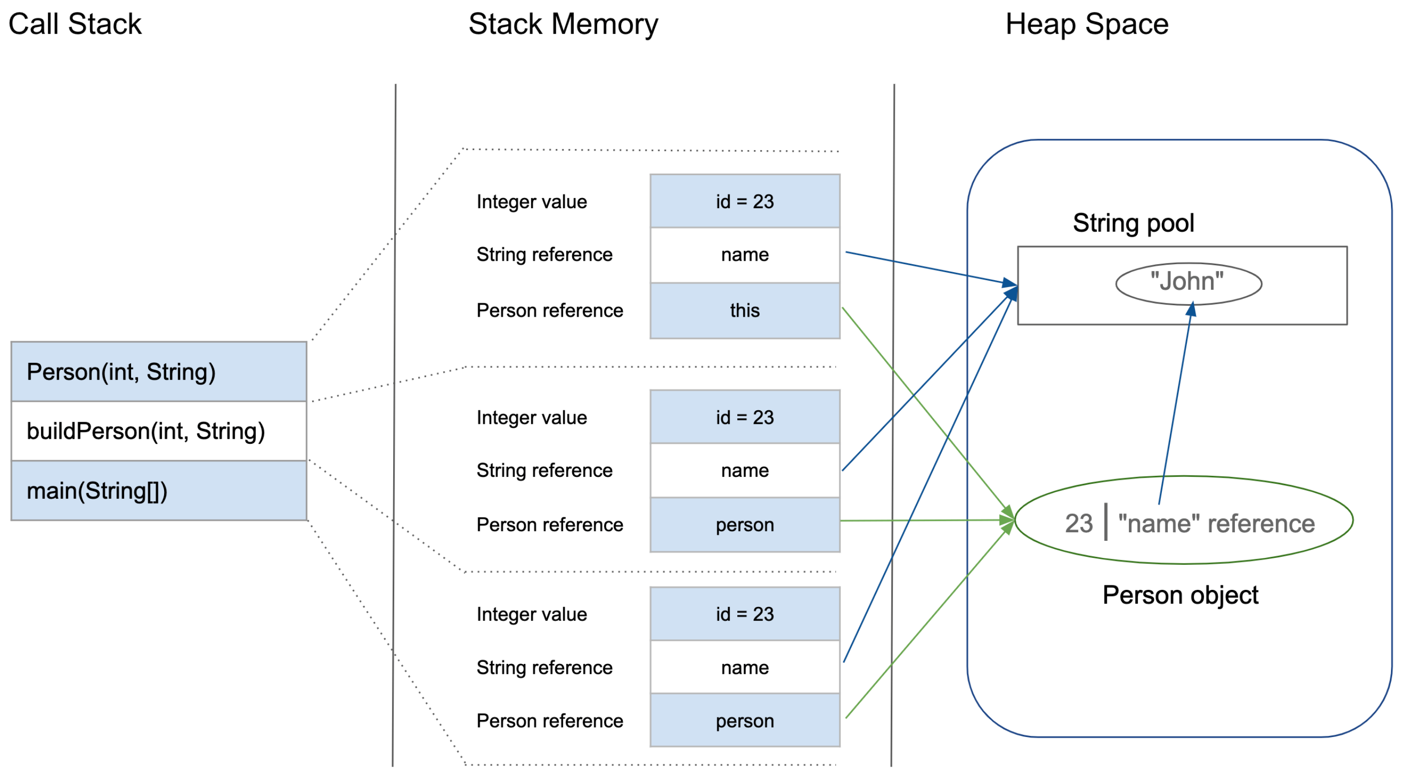
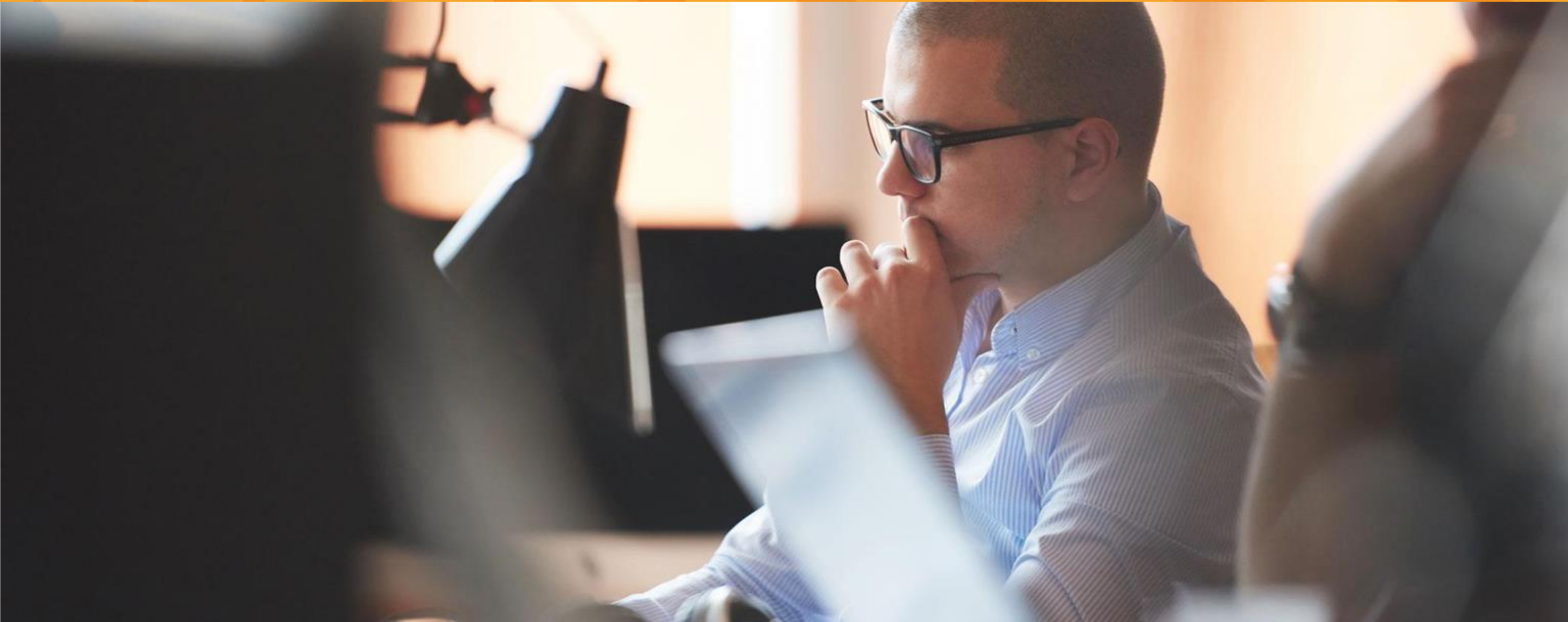


Image source: https://www.baeldung.com/java-stack-heap

REVATURE

Java Casting and Covariance Demo



Java: Exceptions

Events that disrupt the normal flow of our java application

- *Throws* an Exception Object
- Exceptions form a hierarchy
- Two types
  - Checked
  - Unchecked

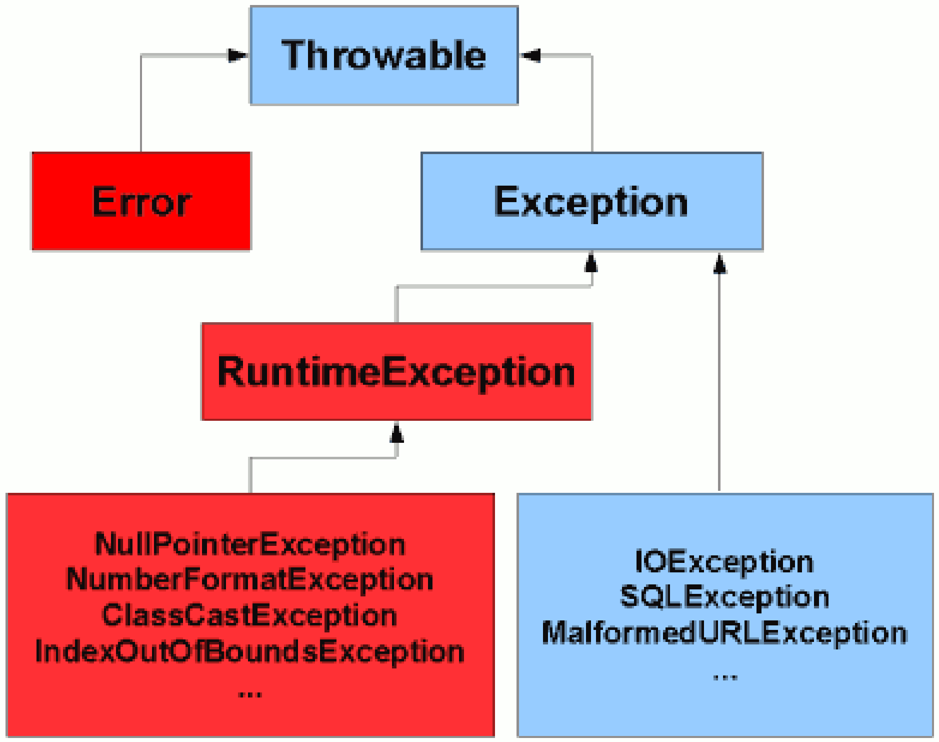
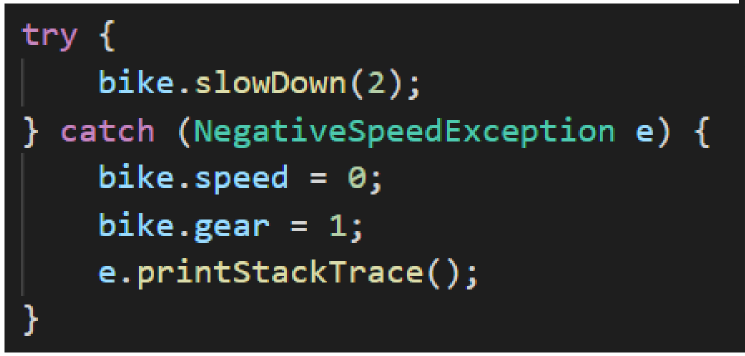


Image Source: https://www.javamex.com/tutorials/exceptions/exceptions_hierarchy.shtml

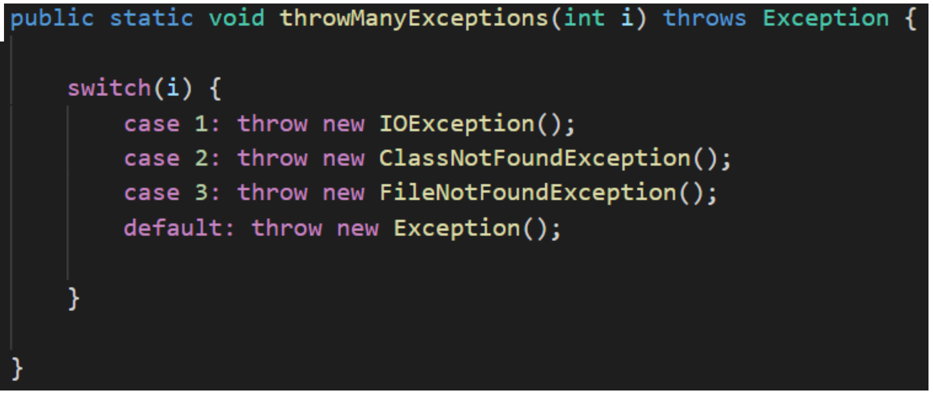
Java: Handling and Declaring Exceptions

Two ways to handle exceptions in java

- Using try/catch blocks
- Try with resources
- Declaring a method can throw an exception
- Cause an exception using the throw keyword



```java
try {
    bike.slowDown(2);
} catch (NegativeSpeedException e) {
    bike.speed = 0;
    bike.gear = 1;
    e.printStackTrace();
}
```

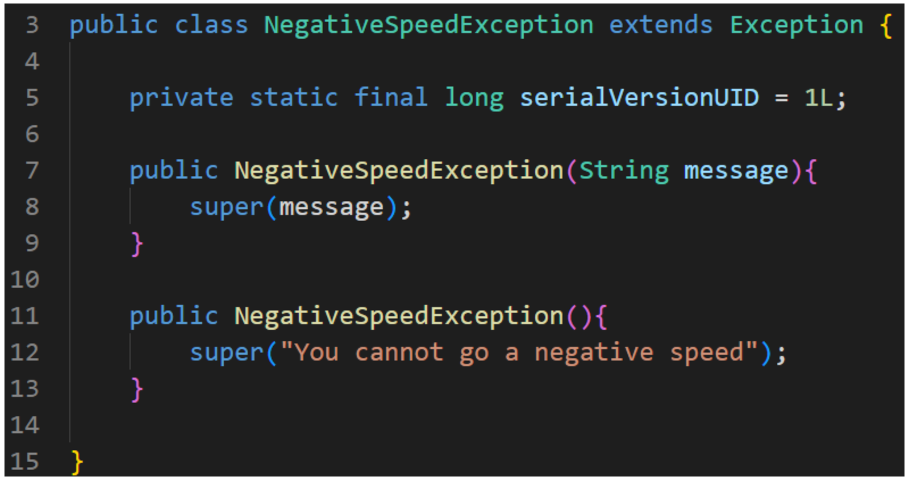


```java
public static void throwManyExceptions(int i) throws Exception {

    switch(i) {
        case 1: throw new IOException();
        case 2: throw new ClassNotFoundException();
        case 3: throw new FileNotFoundException();
        default: throw new Exception();

    }
}
```

Java: Custom Exceptions and Testing

Create custom exceptions by extending Exception or RuntimeException

- Extending Exception creates a checked exception
- Extending RuntimeExceptoin creates an unchecked Exception



```java
public class NegativeSpeedException extends Exception {

    private static final long serialVersionUID = 1L;

    public NegativeSpeedException(String message){
        super(message);
    }

    public NegativeSpeedException(){
        super("You cannot go a negative speed");
    }

}
```

Java Exceptions and Reading Stack Trace Demo

Java: Annotations

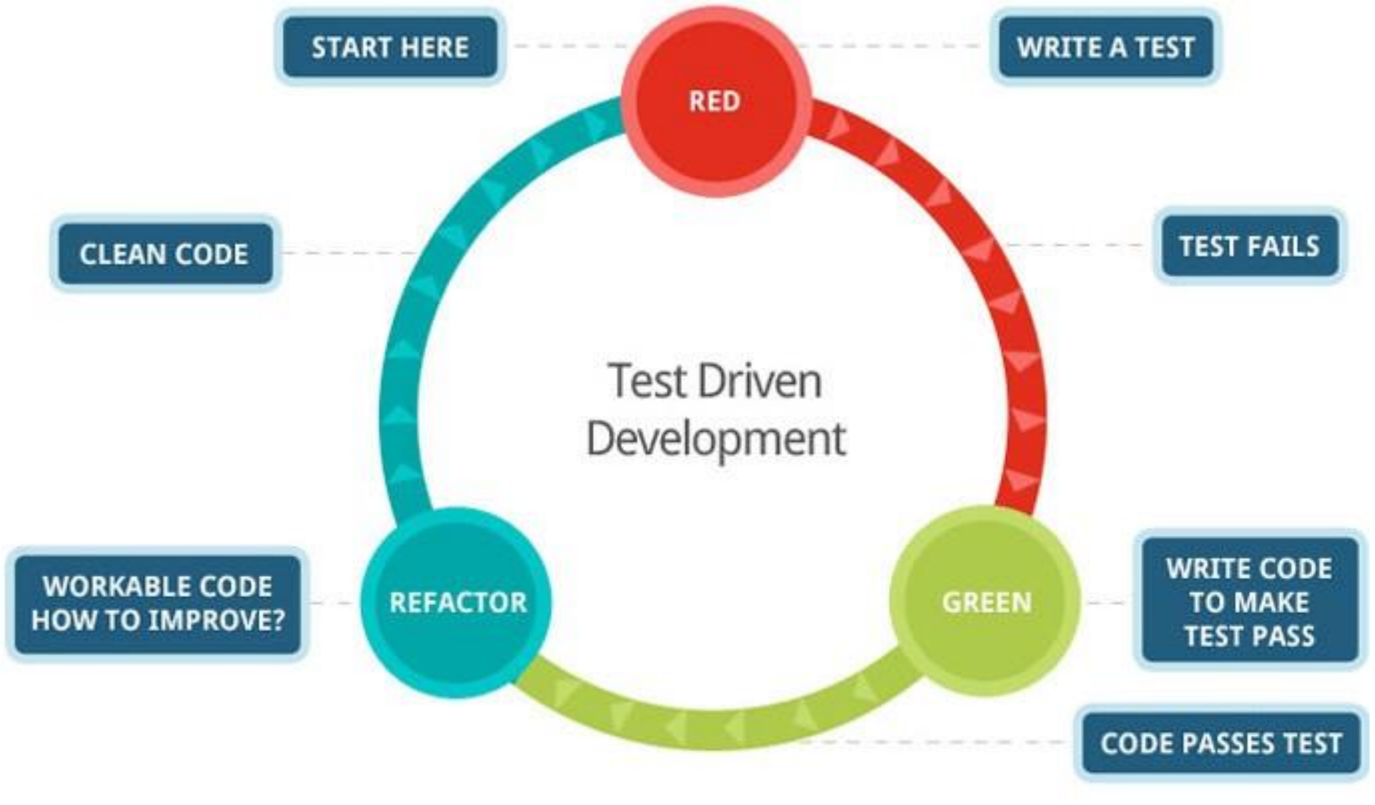
Special constructs in Java with the @ symbol followed by the name of the annotation

- Used to provide metadata to the compiler
- Placed over classes, methods, interfaces, and fields
- Used by java libraries to abstract functionality
- Processed by the Reflections API
- @Override, @Test …

Testing: Test Driven Development

Process of writing tests before our code

- Red/Greed Testing



Testing: Unit Testing

Testing of individual software components

- White box testing
- Helps reduce debugging time
- Increases code coverage
- Increases confidence in making changes to the code base

Testing: JUnit

Java Unit Testing Framework

- Uses annotations to create tests and testing environments
  - @Test, @BeforeTest, @Before, @After, @AfterClass
- Built-in methods to verify the state of your application during the test
  - assertTrue(), assertFalse(), assertEquals(), assertNotEquals(), assertThat()

REVATURE

Java TDD with JUnit Demo