COMP2511

Week 3

THURSDAY 9AM - 12PM (H09A)
FRIDAY 10AM - 1PM (F10A)

Attendance

Ask Questions

Also participation marks

This week

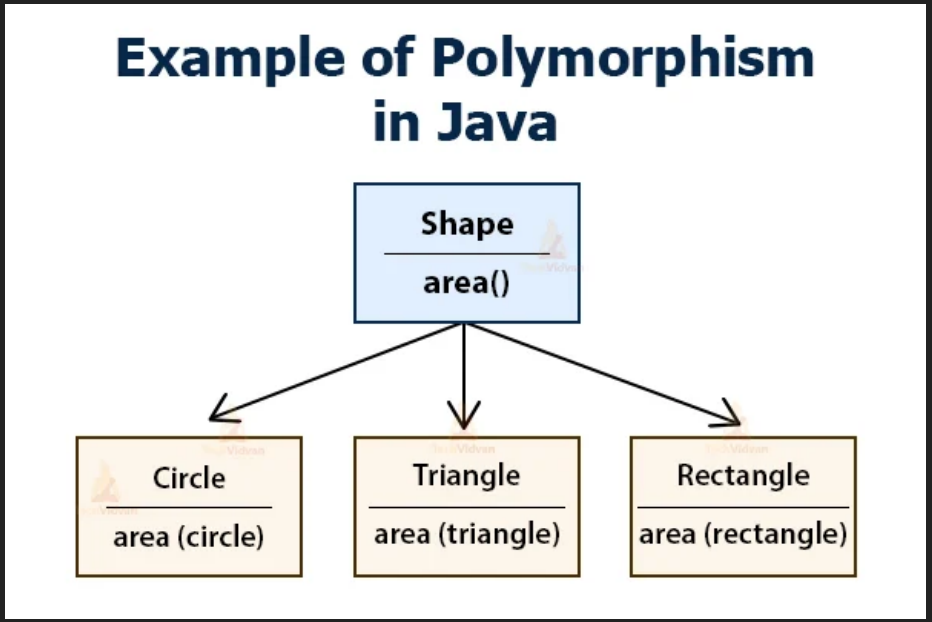
- Design by contract
- **Exceptions**
- **JUnit testing**
- Generics & Collections
- Design Principles

- **Polymorphism/Inheritance**
- **UML**

Polymorphism

What is it?

An object's ability to decide what method to apply to itself, depending on where it is in the inheritance hierarchy, is usually called polymorphism.



Group Task

Code Review - 5 minutes

Code Review

1. Can you override a **static** method?
2. What is the output of **A.f()**?

```java
public class A {
    public static void f() {
        C c = new C();
        c.speak();
        B b = c;
        b.speak();
        b = new B();
        b.speak();
        c.speak();
    }
}


public class B {
    public void speak() {
        System.out.println("moo");
    }
}


public class C extends B {
    public void speak() {
        System.out.println("quack");
    }
}
```

3. What is the output of **D.f()**

```java
public class D {
    public static void f() {
        F f1 = new F();
        F f2 = new F();
        f1.incX();
        f2.incY();
        System.out.println(f1.getX() + " " + f1.getY());
        System.out.println(f2.getX() + " " + f2.getY());
    }
}

public class F {
    private int x;
    private static int y;

    public int getX() {
        return x;
    }

    public int getY() {
        return y;
    }

    public void incX() {
        x++;
    }

    public void incY() {
        y++;
    }
}
```

Code Review

1. Can you override a **static** method?
    - No, since the **static** method is attached to the class, it cannot be overridden
2. What is the output of **A.f()**?
    - quack
    - quack
    - moo
    - quack

```java
public class A {
    public static void f() {
        C c = new C();
        c.speak();
        B b = c;
        b.speak();
        b = new B();
        b.speak();
        c.speak();
    }
}


public class B {
    public void speak() {
        System.out.println("moo");
    }
}


public class C extends B {
    public void speak() {
        System.out.println("quack");
    }
}
```

Code Review

What is the output of **D.f()**

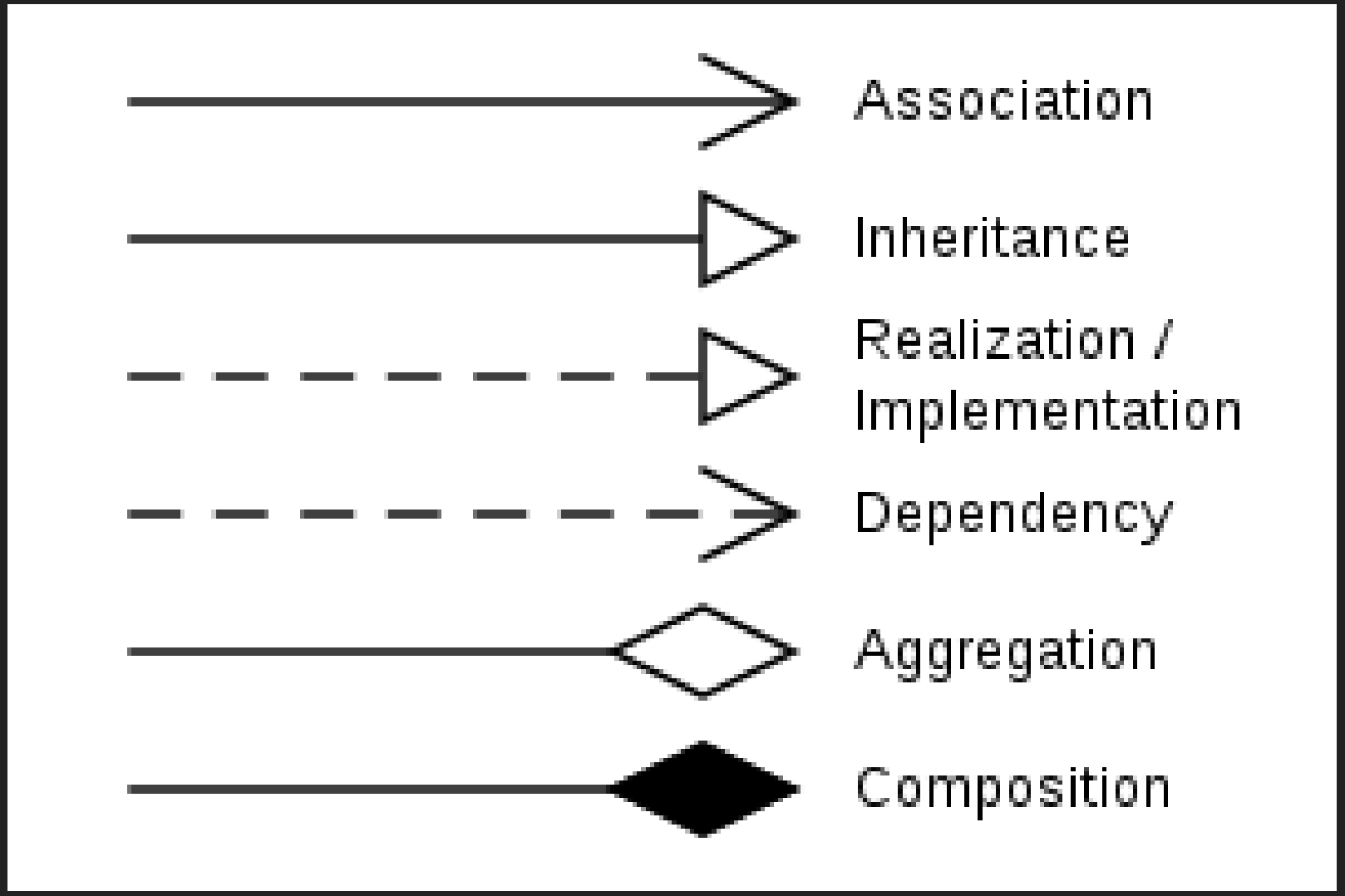
1 1
0 1

```
1  public class D {
2      public static void f() {
3          F f1 = new F();
4          F f2 = new F();
5          f1.incX();
6          f2.incY();
7          System.out.println(f1.getX() + " " + f1.getY());
8          System.out.println(f2.getX() + " " + f2.getY());
9      }
10 }
11
12 public class F {
13     private int x;
14     private static int y;
15
16     public int getX() {
17         return x;
18     }
19
20     public int getY() {
21         return y;
22     }
23
24     public void incX() {
25         x++;
26     }
27
28     public void incY() {
29         y++;
30     }
31 }
```

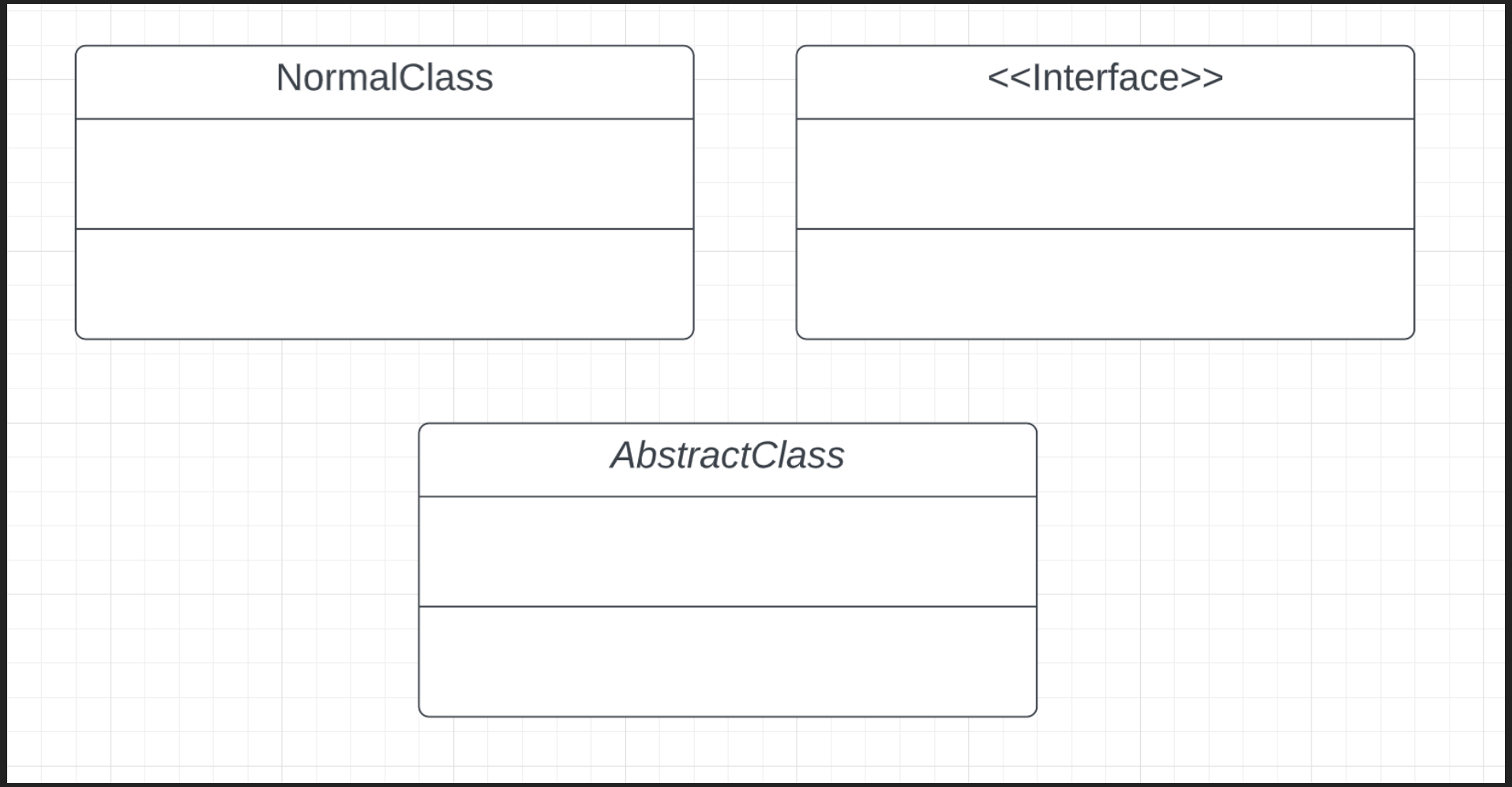
Domain Modelling

UML

UML


Association
Inheritance
Realization /
Implementation
Dependency
Aggregation
Composition

UML



Domain Modelling

Create an OO domain model for a system with the following requirements.

A Car has one or more engines and a producer. The producer is a manufacturing company who has a brand name. Engines are produced by a manufacturer and have a speed. There are only two types of engines within UNSW's cars:

- **Thermal Engines**, which have a default max speed of 114, although they can be produced with a different max speed, and the max speed can change to any value between 100 and 250.
- **Electrical Engines**, which have a default max speed of 180. This is the speed at which they are produced, and the max speed can change to any value that is divisible by 6.

Cars are able to drive to a particular location x, y.

Since UNSW is a world-leader in technology innovation, they want you to be able to model the behaviour of Time Travelling for *any* vehicle, and to model a time travelling car. A vehicle that travels in time *stays in the same location* but travels to a LocalDateTime.

Domain Modelling

Create an OO domain model for a system with the following requirements.

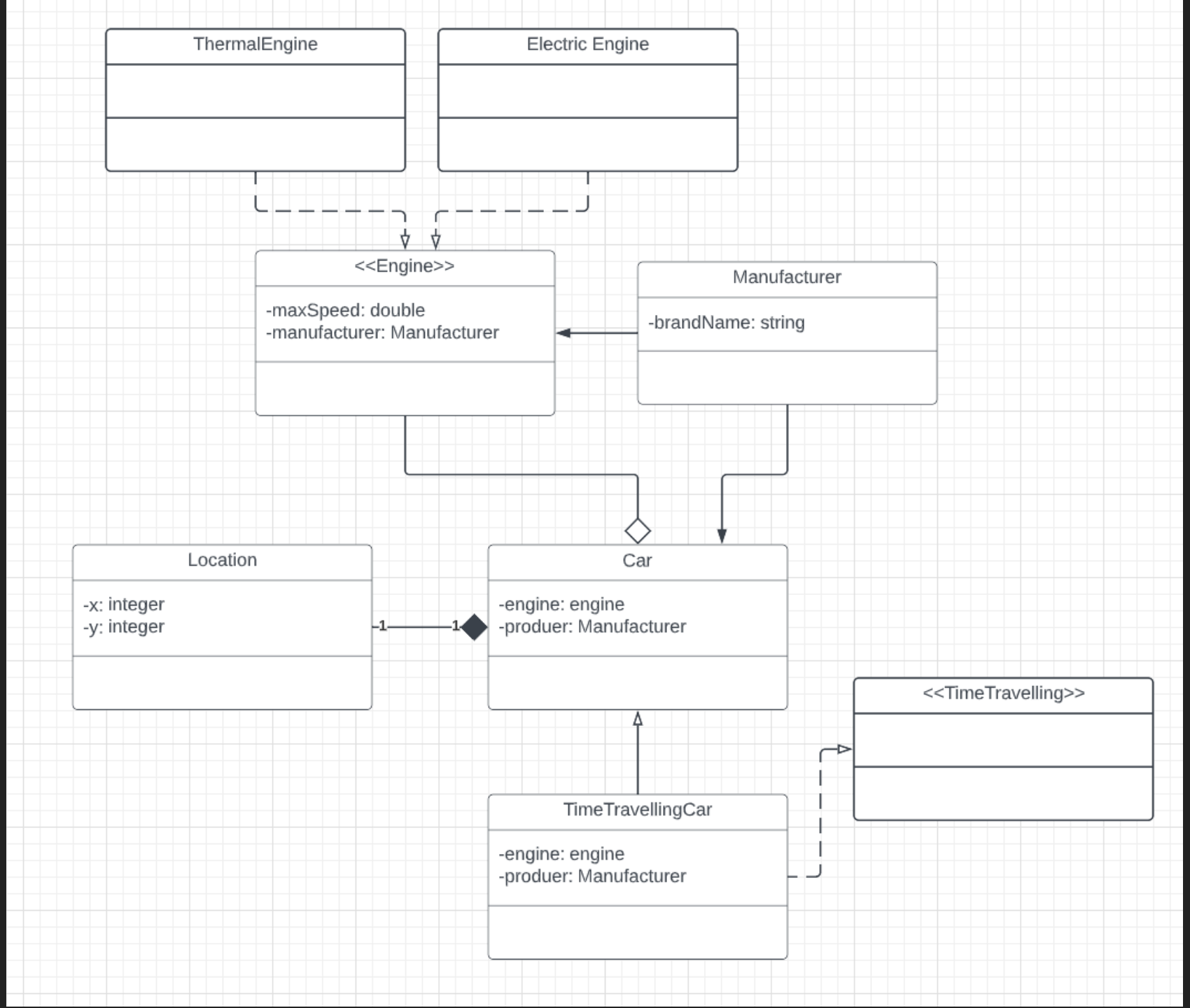
A Car has **one or more engines** and **a producer**. The producer is a manufacturing company who has **a brand name.** Engines are **produced by a manufacturer** and have **a speed**. There are **only two types of engines** within UNSW's cars:

- Thermal Engines: which have a **default max speed of 114**, although they can be produced with a **different max speed**, and the max speed can change to any value between 100 and 250.
- Electrical Engines: which have a **default max speed of 180**. This is the speed at which they are produced, and the max speed can change to any value that is divisible by 6.

Cars are able to drive to a particular location **x, y**.

Since UNSW is a world-leader in technology innovation, they want you to be able to model the behaviour of Time Travelling for *any* vehicle, and to model a time travelling car. A vehicle that travels in time *stays in the same location* but travels to a LocalDateTime.

LucidCharts


ThermalEngine
Electric Engine
<<Engine>>
-maxSpeed: double
-manufacturer: Manufacturer
Manufacturer
-brandName: string
Location
-x: integer
-y: integer
1
1
Car
-engine: engine
-produer: Manufacturer
<<TimeTravelling>>
TimeTravellingCar
-engine: engine
-produer: Manufacturer

Tips for UML Diagrams

- Make it readable
- Use colours to distinguish parts
  - E.g, All satellites are shaded blue...
  - This design pattern is shaded green
- Try making a draft one before you start coding

Code Demo

Wonderous.java

The **Wondrous Sequence** is generated by the simple rule:

- If the current term is even, the next term is half the current term.
- If the current term is odd, the next term is three times the current term, plus 1

```java
package wondrous;

import java.util.ArrayList;
import java.util.List;

public class Wondrous {
    private final int MY_MAGIC_NUMBER = 42;

    public List<Integer> wondrous(int start) {
        int current = start;
        List<Integer> sequence = new ArrayList<Integer>();

        while (current != 1) {
            sequence.add(current);
            if (current % 2 == 0) {
                current /= 2;
            } else {
                current = (current * 3) + 1;
            }
        }

        return sequence;
    }
}
```

JUnit Testing

JUnit Testing

```
You, 3 seconds ago | 1 author (You)
1  package wondrous.test;
2
3  import static org.junit.jupiter.api.Assertions.assertEquals;
4
5  import java.util.ArrayList;
6  import java.util.Arrays;
7  import java.util.List;
8
9  import org.junit.jupiter.api.Test;
10
11 import wondrous.Wondrous;
12
   You, 3 seconds ago | 1 author (You)
13 public class WondrousTest {
14     @Test
15     public void testBasic() {
16         Wondrous w = new Wondrous();
17         List<Integer> expected = new ArrayList<Integer>(Arrays.asList(3, 10, 5, 16, 8, 4, 2, 1));
18
19         assertEquals(expected, w.wondrous(3));
20     }      You, 2 minutes ago • initial commit …
21 }
```

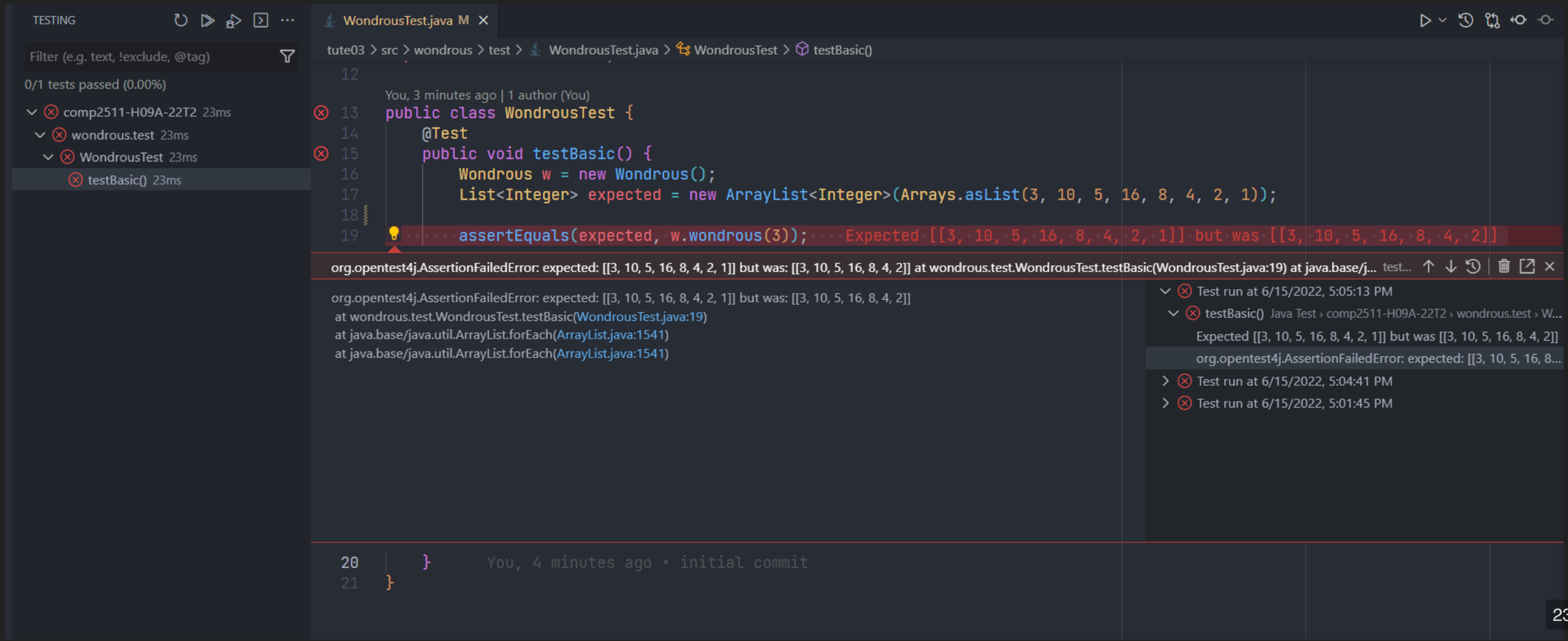
Make sure your VSCode is open in the **correct folder**, otherwise, these bottoms wont appear

JUnit Testing

Make sure your VSCode is open in the **correct folder**, otherwise, these buttons won't appear

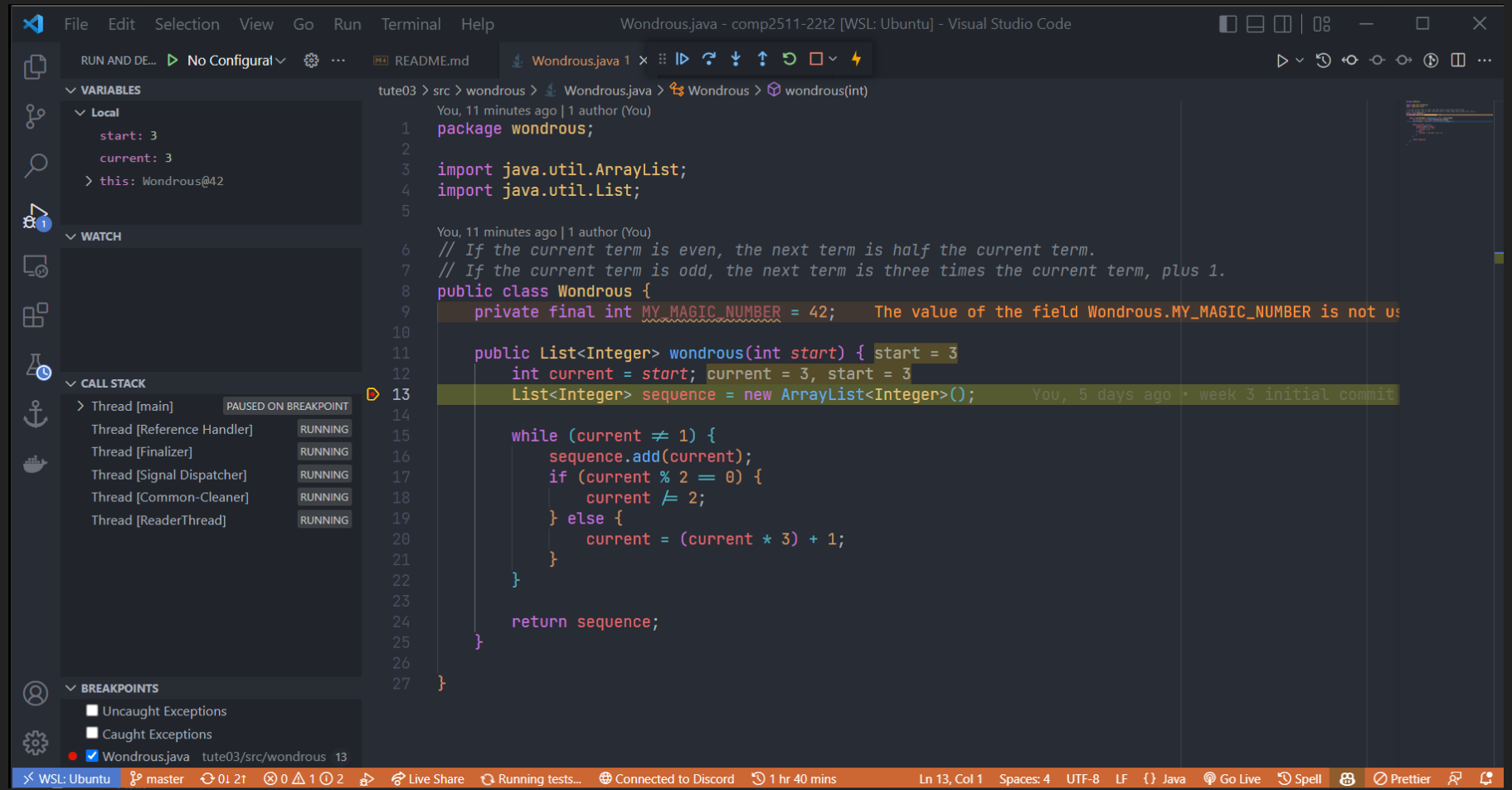
JUnit Testing

- Failed test output
- See past history

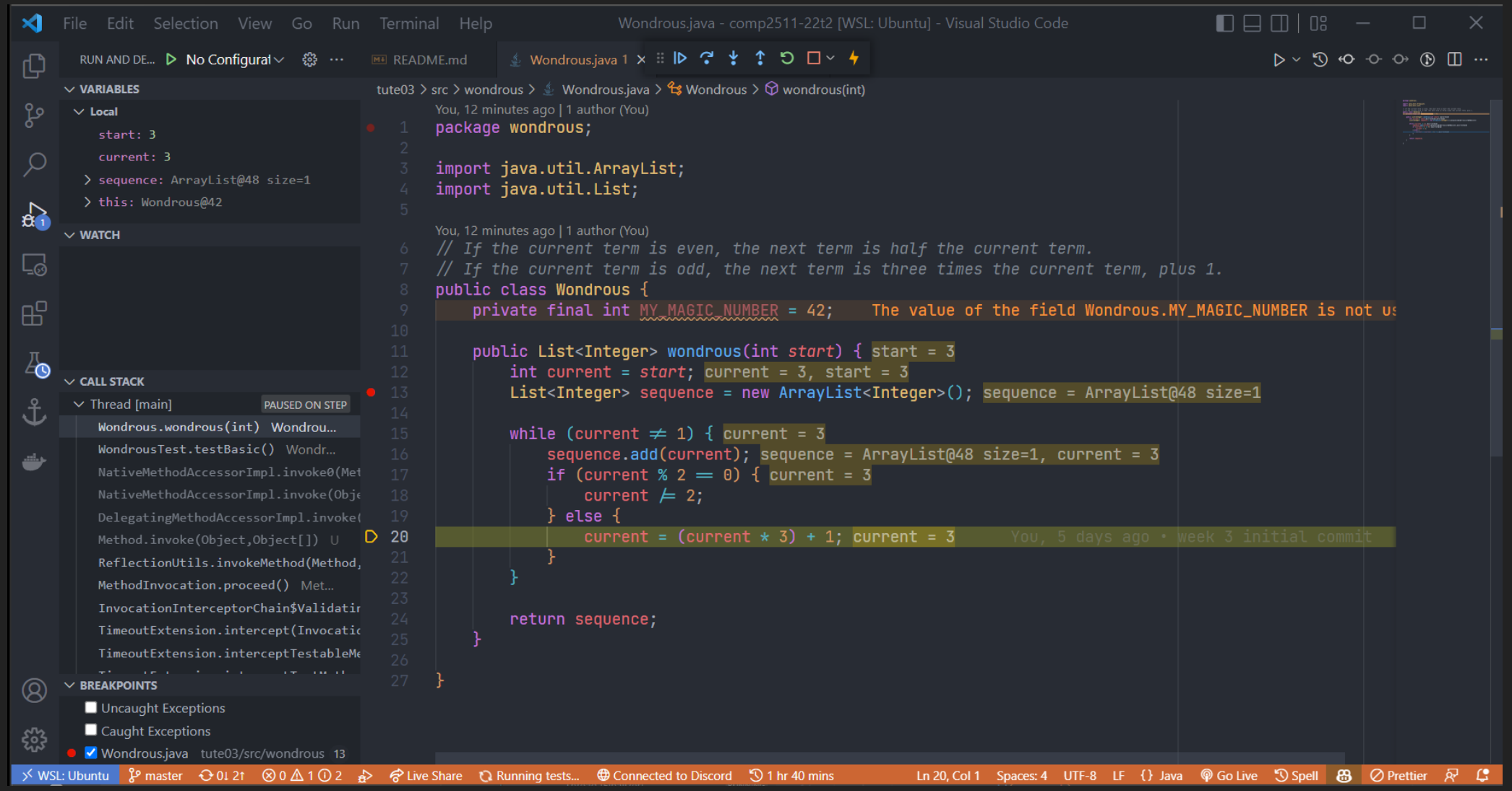


Debug Mode

- Set a breakpoint, run the test in debug mode
- Stops at a breakpoint it encounters
- See variable values, call stack



Debug Mode


File Edit Selection View Go Run Terminal Help
Wondrous.java - comp2511-22t2 [WSL: Ubuntu] - Visual Studio Code
RUN AND DE... No Configurat
README.md
Wondrous.java 1
tute03 > src > wondrous > Wondrous.java > Wondrous > wondrous(int)
VARIABLES
Local
start: 3
current: 3
sequence: ArrayList@48 size=1
this: Wondrous@42
WATCH
CALL STACK
Thread [main] PAUSED ON STEP
Wondrous.wondrous(int) Wondrou...
WondrousTest.testBasic() Wondr...
NativeMethodAccessorImpl.invoke0(Met
NativeMethodAccessorImpl.invoke(Obje
DelegatingMethodAccessorImpl.invoke(
Method.invoke(Object,Object[]) U
ReflectionUtils.invokeMethod(Method,
MethodInvocation.proceed() Met...
InvocationInterceptorChain$Validatir
TimeoutExtension.intercept(Invocatic
TimeoutExtension.interceptTestableMe
BREAKPOINTS
Uncaught Exceptions
Caught Exceptions
Wondrous.java tute03/src/wondrous 13
You, 12 minutes ago | 1 author (You)
package wondrous;
import java.util.ArrayList;
import java.util.List;
You, 12 minutes ago | 1 author (You)
// If the current term is even, the next term is half the current term.
// If the current term is odd, the next term is three times the current term, plus 1.
public class Wondrous {
private final int MY_MAGIC_NUMBER = 42; The value of the field Wondrous.MY_MAGIC_NUMBER is not us
public List<Integer> wondrous(int start) { start = 3
int current = start; current = 3, start = 3
List<Integer> sequence = new ArrayList<Integer>(); sequence = ArrayList@48 size=1
while (current != 1) { current = 3
sequence.add(current); sequence = ArrayList@48 size=1, current = 3
if (current % 2 == 0) { current = 3
current /= 2;
} else {
current = (current * 3) + 1; current = 3 You, 5 days ago • week 3 initial commit
}
}
return sequence;
}
}
WSL: Ubuntu master 0↓ 2↑ ⊗ 0 ⚠ 1 ⓘ 2 Live Share Running tests... Connected to Discord 1 hr 40 mins Ln 20, Col 1 Spaces: 4 UTF-8 LF {} Java Go Live Spell Prettier

Now lets fix it

```java
package wondrous;

import java.util.ArrayList;
import java.util.List;

// If the current term is even, the next term is half the current term.
// If the current term is odd, the next term is three times the current term, plus 1.
public class Wondrous {
    public List<Integer> wondrous(int start) {
        int current = start;
        List<Integer> sequence = new ArrayList<Integer>();

        while (true) {
            sequence.add(current);
            if (current == 1) {
                break;
            } else if (current % 2 == 0) {
                // Even
                current /= 2;
            } else {
                // odd
                current = (current * 3) + 1;
            }
        }
        return sequence;
    }
}
```

Exceptions

Exceptions

What are they?

- An exception is an event, which occurs during the execution of a problem, that **disrupt the normal flow of the program's instructions**
- When error occurs, an exception object is created and given to the runtime system. This is called **throwing an exception**
- The runtime system searches the call stack for a method that contains a block of code that can handle the exception
- The exception handler chosen is said to catch the exception

Exceptions

Checked vs Unchecked

- Checked: Must be checked, will result in compilation error if not handled
  - Any class that inherits from `Exception` is a checked exception.
  - E.g., IOException, SQLException
- Unchecked: Genuine errors that occur at run time
  - Any class that inherits from `RuntimeException` is unchecked
  - E.g., ArrayIndexOutOfBoundsExceptions, ArithmeticException

Exceptions - Wonderous

How can we use it in Wonderous to make the code better?
Checked or Unchecked?

Exceptions - Wonderous

How can we use it in Wonderous to make the code better?

```java
package wondrous;

import java.util.ArrayList;
import java.util.List;

// If the current term is even, the next term is half the current term.
// If the current term is odd, the next term is three times the current term, plus 1.
public class Wondrous {
    public List<Integer> wondrous(int start) {
        int current = start;
        List<Integer> sequence = new ArrayList<Integer>();

        if (start < 1) {
            throw new IllegalArgumentException("wondrous start must be >= 1");
        }

        while (true) {
            sequence.add(current);
            if (current == 1) {
                break;
            } else if (current % 2 == 0) {
                // Even
                current /= 2;
            } else {
                // odd
                current = (current * 3) + 1;
            }
        }
        return sequence;
    }
}
```

JUnit Testing

Lets write some more tests

```java
package wondrous.test;

import static org.junit.Assert.assertThrows;
import static org.junit.jupiter.api.Assertions.assertEquals;

import java.util.ArrayList;
import java.util.Arrays;
import java.util.List;

import org.junit.jupiter.api.Test;

import wondrous.Wondrous;

public class WondrousTest {
    @Test
    public void testBasic() {
        Wondrous w = new Wondrous();
        List<Integer> expected = new ArrayList<Integer>(Arrays.asList(3, 10, 5, 16, 8, 4, 2, 1));

        assertEquals(expected, w.wondrous(3));
    }

    @Test
    public void testOne() {
        Wondrous w = new Wondrous();
        List<Integer> expected = new ArrayList<Integer>(Arrays.asList(1));
        assertEquals(expected, w.wondrous(1));
    }

    @Test
    public void testInvalid() {
        Wondrous w = new Wondrous();

        assertThrows(IllegalArgumentException.class, () -> {
            w.wondrous(0);
        });
    }
}
```

Feedback

https://forms.gle/fZDe2zhbo52UNnwh7