

COMP2511

23T1 Week 3

WEDNESDAY 1PM - 4PM (W13B)

FRIDAY 11AM - 2PM (F11A)

Slides by Alvin Cherk (z5311001)

Today

- Design by contract
 - **Exceptions**
 - **JUnit testing**
 - Generics & Collections
 - Design Principles
-
- **Polymorphism/Inheritance**
 - **UML**

Important Notice

- Assignment is out, please start early
- There is an assignment viva (discussion with me) after its due in week 7/8. This is to make it easier for me to mark your design decision & to prevent plagiarism
- Don't plagiarise, don't use code synthesis tools like ChatGPT
- Go to help sessions if you need help (they get busier towards the due date)
- No auto generated UMLs of any sort. You will get 0. If you use programs that generate it based on text, please commit this text as a file in your repo.
- Give me feedback, the tutorial is for you. I want to know what I can do better

<https://forms.gle/R4sMTTQzPC4vqXSN8>

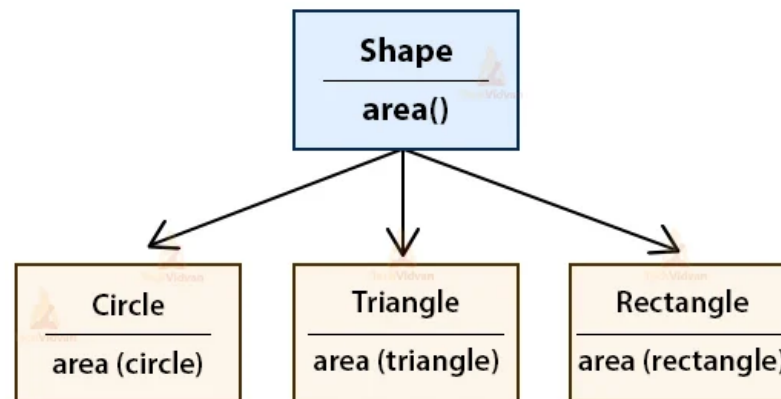
Polymorphism

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.

Example of Polymorphism in Java



Group Task

Code Review - 5 minutes

Code Review

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

```
1 public class A {
2     public static void f() {
3         C c = new C();
4         c.speak();
5         B b = c;
6         b.speak();
7         b = new B();
8         b.speak();
9         c.speak();
10    }
11 }
12
13
14 public class B {
15     public void speak() {
16         System.out.println("moo");
17     }
18 }
19
20
21 public class C extends B {
22     public void speak() {
23         System.out.println("quack");
24     }
25 }
```

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

```
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 }
```

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

```
1 public class A {
2     public static void f() {
3         C c = new C();
4         c.speak();
5         B b = c;
6         b.speak();
7         b = new B();
8         b.speak();
9         c.speak();
10    }
11 }
12
13
14 public class B {
15     public void speak() {
16         System.out.println("moo");
17     }
18 }
19
20
21 public class C extends B {
22     public void speak() {
23         System.out.println("quack");
24     }
25 }
```


Code Review

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

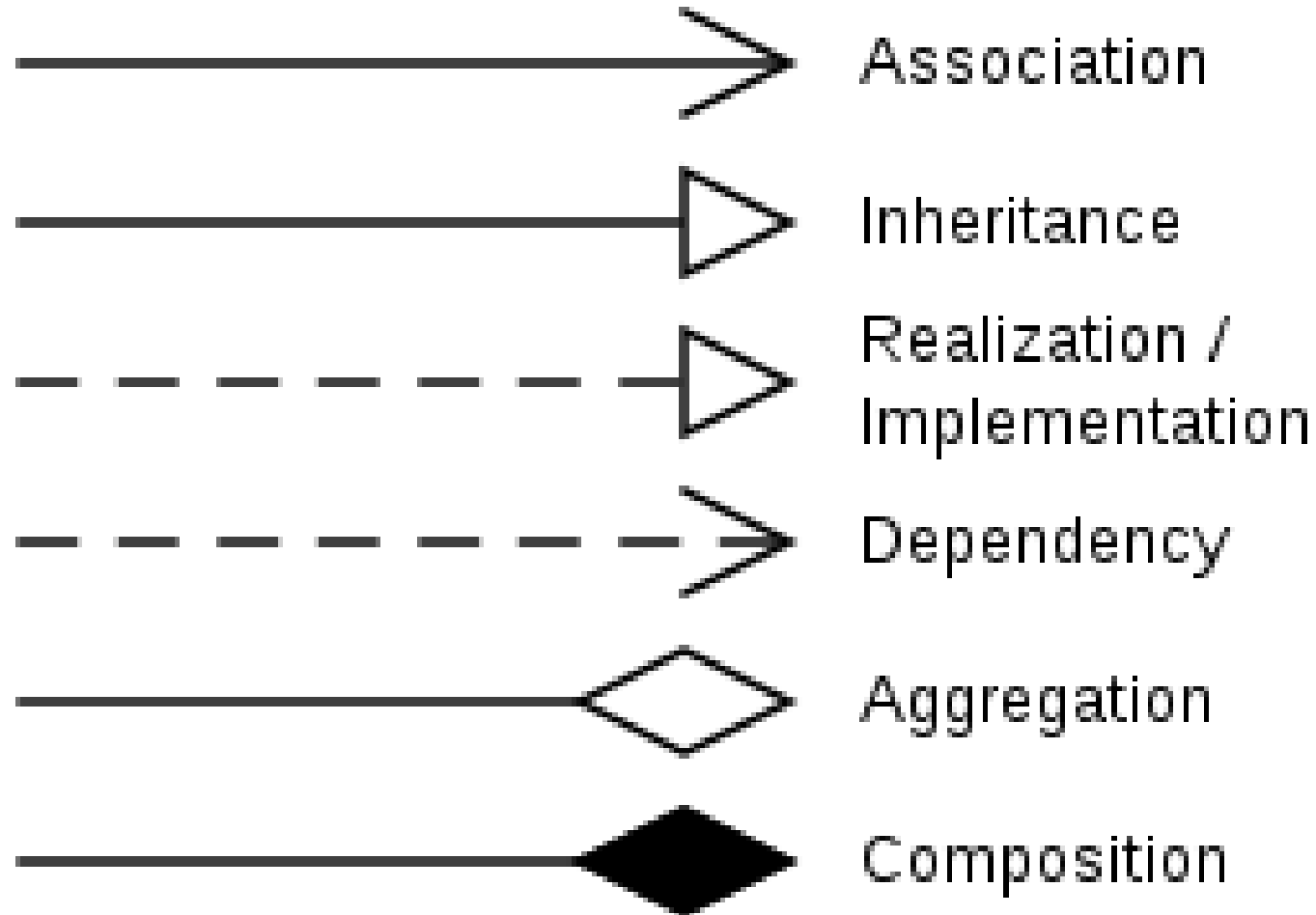
1 1
0 1

```
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

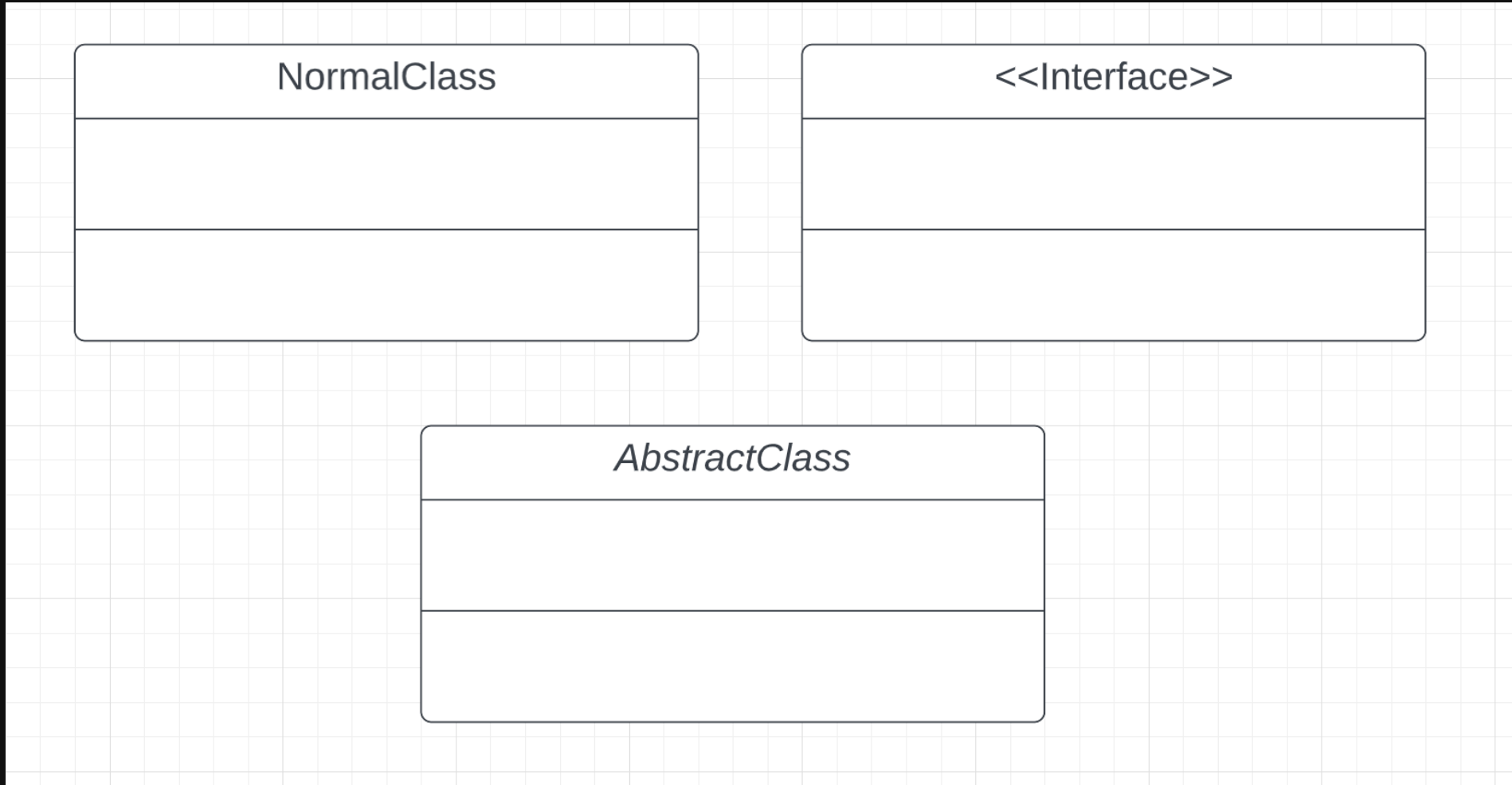
UML



UML

Composition is a specialized form of aggregation. In composition, if the parent object is destroyed, then the child objects also cease to exist. Composition is actually a strong type of aggregation and is sometimes referred to as a "death" relationship.

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.

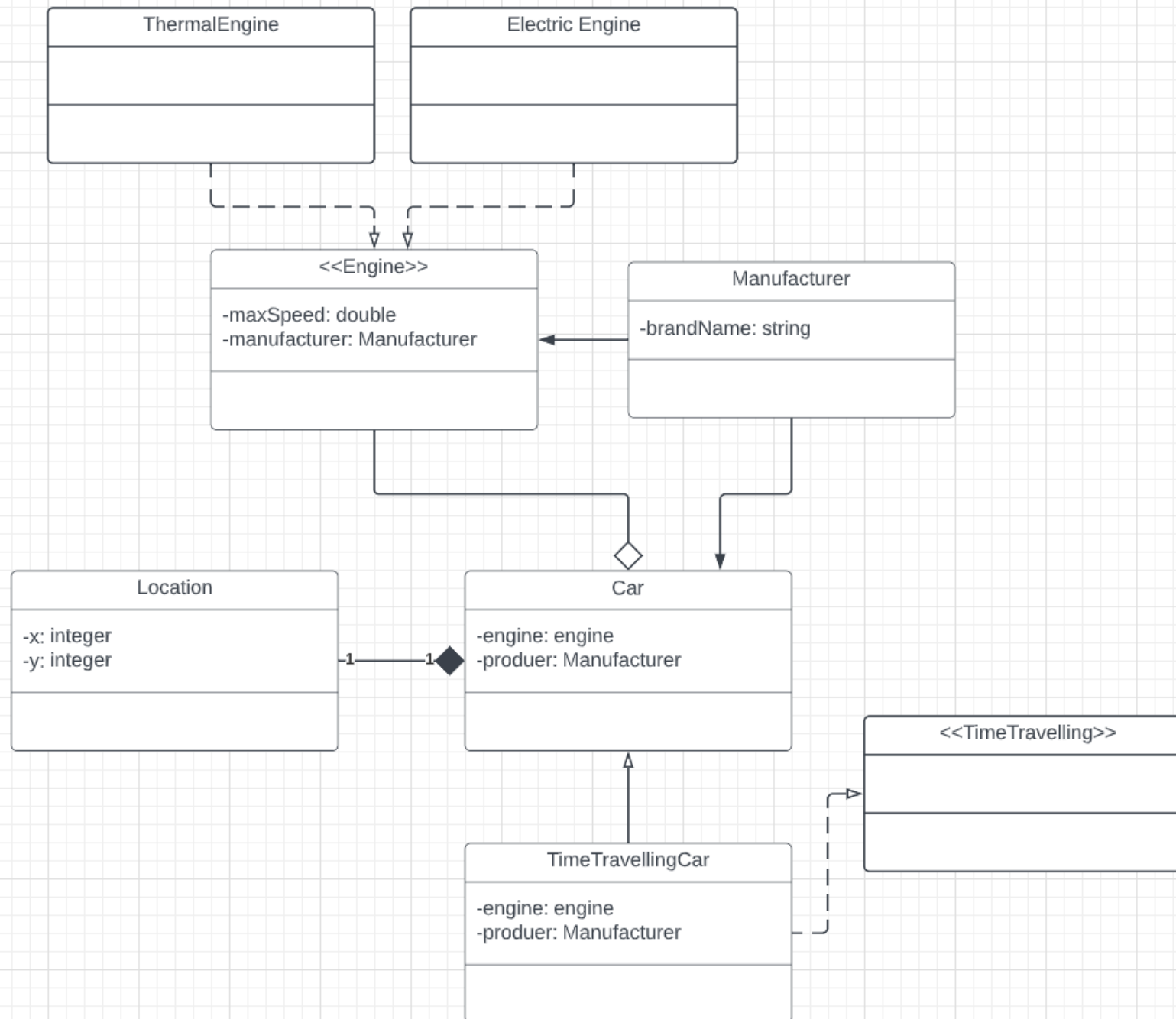
Cars are able to drive to a particular location **x, y**.

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

A generally good diagram maker software that is free*

*if you sign up with an .edu account



Tips for UML Diagrams

- Make it readable (for me please:))
- 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 your assignments

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

```
1 package wondrous;
2
3 import java.util.ArrayList;
4 import java.util.List;
5
6 public class Wondrous {
7     private final int MY_MAGIC_NUMBER = 42;
8
9     public List<Integer> wondrous(int start) {
10         int current = start;
11         List<Integer> sequence = new ArrayList<Integer>();
12
13         while (current != 1) {
14             sequence.add(current);
15             if (current % 2 == 0) {
16                 current /= 2;
17             } else {
18                 current = (current * 3) + 1;
19             }
20         }
21
22         return sequence;
23     }
24 }
```

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
13  You, 3 seconds ago | 1 author (You)
14  public class WondrousTest {
15      @Test
16      public void testBasic() {
17          Wondrous w = new Wondrous();
18          List<Integer> expected = new ArrayList<Integer>(Arrays.asList(3, 10, 5, 16, 8, 4, 2, 1));
19
20          assertEquals(expected, w.wondrous(3));
21      }
22  }
```

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

JUnit Testing

```
You, 2 minutes 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
13 You, 2 minutes ago | 1 author (You)
14 public class WondrousTest {
15     @Test
16     public void testBasic() {
17         // = new Wondrous();
18         Integer[] expected = new Integer[]{3, 10, 5, 16, 8, 4, 2, 1};
19         assertEquals(expected, w.wondrous(3));
20     }
21 }
minutes ago • initial commit ...
```

A screenshot of the Visual Studio Code editor interface. The background is dark with light-colored text. The code is written in Java and uses syntax highlighting. A context menu is open over the test method on line 16, showing options like 'Run Test', 'Debug Test', 'Peek Error', and 'Reveal in Test Explorer'. The menu has a light gray background and dark text. The code in the background is for a JUnit test class named 'WondrousTest' that tests a 'w.wondrous(3)' method against an expected array of integers. The code is partially obscured by the context menu, but the test method 'testBasic' is clearly visible. The context menu is positioned on the left side of the editor, and the code continues on the right side. The overall layout is clean and professional, typical of a development environment.

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

JUnit Testing

- Failed test output
- See past history

TESTING [refresh] [run] [debug] [compare] [more]

Filter (e.g. text, !exclude, @tag) [filter icon]

0/1 tests passed (0.00%)

- ✖ comp25111-H09A-22T2 23ms
- ✖ wondrous.test 23ms
 - ✖ WondrousTest 23ms
 - ✖ testBasic() 23ms

tute03 > src > wondrous > test > WondrousTest.java > WondrousTest > testBasic()

```
12
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)); Expected: [[3, 10, 5, 16, 8, 4, 2, 1]] but was: [[3, 10, 5, 16, 8, 4, 2]]

```

org.opentest4j.AssertionFailedError: expected: [[3, 10, 5, 16, 8, 4, 2, 1]] but was: [[3, 10, 5, 16, 8, 4, 2]] at wondrous.test.WondrousTest.testBasic(WondrousTest.java:19) at java.base/j... test... [up] [down] [refresh] [delete] [share] [close]

org.opentest4j.AssertionFailedError: expected: [[3, 10, 5, 16, 8, 4, 2, 1]] but was: [[3, 10, 5, 16, 8, 4, 2]]
at wondrous.test.WondrousTest.testBasic(WondrousTest.java:19)
at java.base/java.util.ArrayList.forEach(ArrayList.java:1541)
at java.base/java.util.ArrayList.forEach(ArrayList.java:1541)

20 } You, 4 minutes ago • initial commit
21 }

Test run at 6/15/2022, 5:05:13 PM

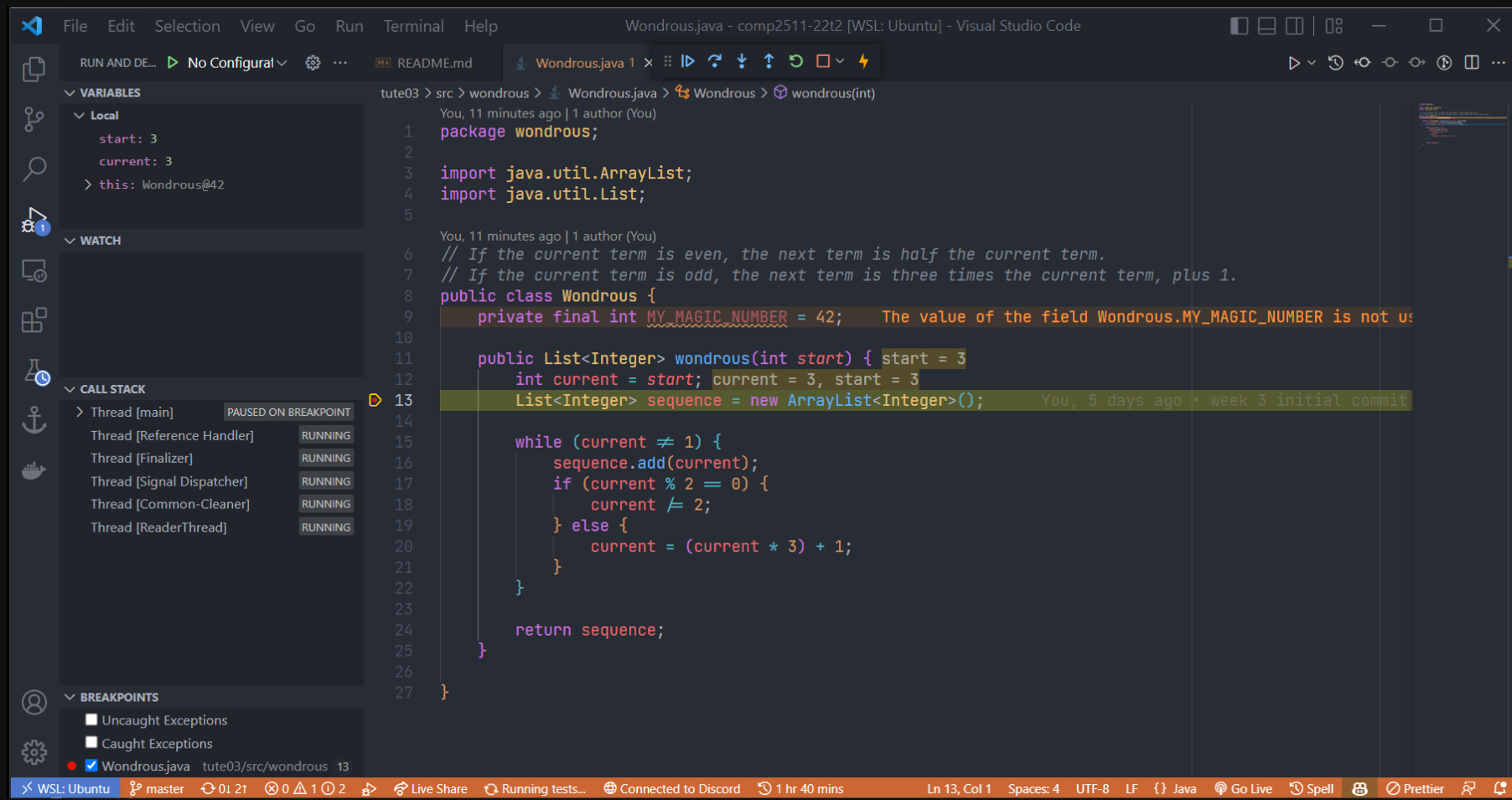
✖ testBasic() Java Test > comp25111-H09A-22T2 > wondrous.test > W...
Expected [[3, 10, 5, 16, 8, 4, 2, 1]] but was [[3, 10, 5, 16, 8, 4, 2]]
org.opentest4j.AssertionFailedError: expected: [[3, 10, 5, 16, 8...

> ✖ Test run at 6/15/2022, 5:04:41 PM

> ✖ Test run at 6/15/2022, 5:01:45 PM

Debug Mode

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



Debug Mode

The screenshot shows the Visual Studio Code interface with a Java file named `Wondrous.java` open. The file is located at `src > wondrous > Wondrous.java`. The code defines a `Wondrous` class with a `wondrous(int start)` method that generates a sequence of numbers based on the following logic:

- 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.

The method starts with `start = 3` and `current = 3`. It uses a `while` loop to generate the sequence until `current` is 1. The sequence is stored in a `ArrayList` named `sequence`.

The left sidebar shows the **DEBUG CONSOLE** with the following sections:

- VARIABLES**:
 - Local**:
 - `start`: 3
 - `current`: 3
 - `sequence`: `ArrayList@48 size=1`
 - `this`: `Wondrous@42`
- WATCH**: (Empty)
- CALL STACK**:
 - Thread [main]**:
 - `Wondrous.wondrous(int) Wondrou...` (PAUSED ON STEP)
 - `WondrousTest.testBasic() Wondr...`
 - `NativeMethodAccessorImpl.invoke0(Met...`
 - `NativeMethodAccessorImpl.invoke(Obj...`
 - `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

The bottom status bar shows the following information:

- WSL: Ubuntu
- master
- 0:21
- 0:0
- 1:02
- 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

```

1 package wondrous;
2
3 import java.util.ArrayList;
4 import java.util.List;
5
6 // If the current term is even, the next term is half the current term.
7 // If the current term is odd, the next term is three times the current term, plus 1.
8 public class Wondrous {
9     public List<Integer> wondrous(int start) {
10         int current = start;
11         List<Integer> sequence = new ArrayList<Integer>();
12
13         while (true) {
14             sequence.add(current);
15             if (current == 1) {
16                 break;
17             } else if (current % 2 == 0) {
18                 // Even
19                 current /= 2;
20             } else {
21                 // odd
22                 current = (current * 3) + 1;
23             }
24         }
25         return sequence;
26     }
27 }

```

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., `ArrayIndexOutOfBoundsException`, `ArithmeticException`

Exceptions - Wonderous

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

Exceptions - Wonderful

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

```
1 package wondrous;
2
3 import java.util.ArrayList;
4 import java.util.List;
5
6 // If the current term is even, the next term is half the current term.
7 // If the current term is odd, the next term is three times the current term, plus 1.
8 public class Wondrous {
9     public List<Integer> wondrous(int start) {
10         int current = start;
11         List<Integer> sequence = new ArrayList<Integer>();
12
13         if (start < 1) {
14             throw new IllegalArgumentException("wondrous start must be >= 1");
15         }
16
17         while (true) {
18             sequence.add(current);
19             if (current == 1) {
20                 break;
21             } else if (current % 2 == 0) {
22                 // Even
23                 current /= 2;
24             } else {
25                 // odd
26                 current = (current * 3) + 1;
27             }
28         }
29         return sequence;
30     }
31 }
```

JUnit Testing

Lets write some more tests

```

1 package wondrous.test;
2
3 import static org.junit.Assert.assertThrows;
4 import static org.junit.jupiter.api.Assertions.assertEquals;
5
6 import java.util.ArrayList;
7 import java.util.Arrays;
8 import java.util.List;
9
10 import org.junit.jupiter.api.Test;
11
12 import wondrous.Wondrous;
13
14 public class WondrousTest {
15     @Test
16     public void testBasic() {
17         Wondrous w = new Wondrous();
18         List<Integer> expected = new ArrayList<Integer>(Arrays.asList(3, 10, 5, 16, 8, 4, 2, 1));
19
20         assertEquals(expected, w.wondrous(3));
21     }
22
23     @Test
24     public void testOne() {
25         Wondrous w = new Wondrous();
26         List<Integer> expected = new ArrayList<Integer>(Arrays.asList(1));
27         assertEquals(expected, w.wondrous(1));
28     }
29
30     @Test
31     public void testInvalid() {
32         Wondrous w = new Wondrous();
33
34         assertThrows(IllegalArgumentException.class, () -> {
35             w.wondrous(0);
36         });
37     }
38 }

```

Feedback



<https://forms.gle/R4sMTTQzPC4vqXSN8>