Topics for this Lecture

- Understand how object orientation features affect software testing
- The difference between procedural and object oriented testing

What is OO programming?.

- Grady Booch [p. 44] defines it as the following:

"Object-oriented programming is a method of implementation in which programs are organized as cooperative collections of objects, each of which represents an instance of some class, and whose classes are all members of a hierarchy of classes united via inheritance relationships."

- There are three important parts to this definition: object-oriented programming **(1)** uses objects; **(2)** each object is an instance of a class; and **(3)** classes are related to one another via inheritance relationships.

Procedural vs OO Programming

- Less complexity in Procedural programs
  - unit = single program, function, or procedure
- More complexity in OO
  - unit = class or (small) cluster of strongly related classes
    - dealing with single methods separately is usually too expensive, so methods are usually tested in the context of the class they belong to
- Less problems in Procedural programs
  - Procedural programs testing typically require generating input of values for arguments of procedure or function under test as well as global variables.
- More problems in OO
  - object-oriented testing not only requires generating primitive inputs, but also requires generating desirable object instances for both receiver object and arguments of method under test.
    - The state of an object is modified by method calls.

Class Testing Basics

- Approach:
  - Identify the class under test (CUT)
  - Identify all methods, constructors, and fields applicable to a class (e.g., public methods)
  - Identify the dependencies (parameters)
  - Generate equivalent and non-equivalent sequences of method invocations
- **In principle:** *"Two states are equivalent if all possible sequences of methods starting from those states produce the same results"*
  - For example, method pop() of class Stack in the Java standard library is equivalent to the method sequence remove(size()-1).

Challenges of OO Testing

- **Encapsulation:**
  - The object's attributes can only be accessed through a sequence of specific method calls.
- **Inheritance:**
  - Methods inherited from super-classes should be retested in the context of sub-classes
- **Polymorphism:**
  - It introduces undecidability in program testing because the exact implementation of a method cannot be known until runtime
- **Generics:**
  - It allow programmers to parameterize classes and methods with types, such that the method can be called for different types. While generics prevent code duplication, this feature makes it difficult to know what is the exact type for generic type parameters

Stack Example

```
import java.util.NoSuchElementException;
public class Stack {
     private static int MAX_ELEMENTS=10 ;
     private int[] values= new int[MAX_ELEMENTS];
     private int size=0;
     public Stack() {
     }
     public void push(int x) {
          if (isFull())
               throw new IllegalStateException("Cannot add to full stack");
          else
               values[size++] = (Integer) x;
          }
     private boolean isFull() {
          if (size >= MAX_ELEMENTS)
               return true;
          else
               return false;
     }
     public int pop() {
          if (isEmpty())
               throw new NoSuchElementException("Cannot pop from empty stack");
           else
               return values[--size];
     }
     private boolean isEmpty() {
          if (size == 0)
               return true;
          else
               return false;
     }
     public int top() {
          if (isEmpty())
               throw new NoSuchElementException("Cannot pop from empty stack");
          else
               return values[size - 1];
     }
}
```

How many test cases needed to cover all feasible branches?

Oregon State University

Stack Example

```
import java.util.NoSuchElementException;
public class Stack {
     private static int MAX_ELEMENTS=3 ;
     private int[] values= new int[MAX_ELEMENTS];
     private int size=0;
     public Stack() {
     }
     public void push(int x) {
          if (isFull())
               throw new IllegalStateException("Cannot add to full stack");
          else
               values[size++] = (Integer) x;
          }
     private boolean isFull() {
          if (size >= MAX_ELEMENTS)
               return true;
          else
               return false;
     }
     public int pop() {
          if (isEmpty())
               throw new NoSuchElementException("Cannot pop from empty stack");
           else
               return values[--size];
     }
     private boolean isEmpty() {
          if (size == 0)
               return true;
          else
               return false;
     }
     public int top() {
          if (isEmpty())
               throw new NoSuchElementException("Cannot pop from empty stack");
          else
               return values[size - 1];
     }
}
```

How many test cases needed to cover all feasible branches?

We need to cover 5 decisions (10 branches)

Oregon State University

Stack Example

```
import java.util.NoSuchElementException;
public class Stack {
     private static int MAX_ELEMENTS=3 ;
     private int[] values= new int[MAX_ELEMENTS];
     private int size=0;
     public Stack() {
     }
     public void push(int x) {
          if (isFull())
               throw new IllegalStateException("Cannot add to full stack");
          else
               values[size++] = (Integer) x;
          }
     private boolean isFull() {
          if (size >= MAX_ELEMENTS)
               return true;
          else
               return false;
     }
     public int pop() {
          if (isEmpty())
               throw new NoSuchElementException("Cannot pop from empty stack");
           else
               return values[--size];
     }
     private boolean isEmpty() {
          if (size == 0)
               return true;
          else
               return false;
     }
     public int top() {
          if (isEmpty())
               throw new NoSuchElementException("Cannot pop from empty stack");
          else
               return values[size - 1];
     }
}
```

How many test cases needed to cover all feasible branches?

We need to cover 5 decisions (10 branches)

We have only 3 public methods that we can call: push (int x), pop(), and top()

Oregon State University

Stack Example

```
import java.util.NoSuchElementException;
public class Stack {
     private static int MAX_ELEMENTS=3 ;
     private int[] values= new int[MAX_ELEMENTS];
     private int size=0;
     public Stack() {
     }
     public void push(int x) {
          if (isFull())
               throw new IllegalStateExcep
          else
               values[size++] = (Integer) x
          }
     private boolean isFull() {
          if (size >= MAX_ELEMENTS)
               return true;
          else
               return false;
     }
     public int pop() {
          if (isEmpty())
               throw new NoSuchElementExce
           else
               return values[--size];
     }
     private boolean isEmpty() {
          if (size == 0)
               return true;
          else
               return false;
     }
     public int top() {
          if (isEmpty())
               throw new NoSuchElementExc
          else
               return values[size - 1];
     }
}
```

How many test cases needed to cover all feasible branches?

Test Case#1	Test Case#2

Stack stack0 = new Stack();
stack0.push((16));
stack0.push((16));
stack0.push((16));
try {
stack0.push((16));
}
catch(EmptyStackException e)
{
}
	Stack stack0 = new Stack();
	stack0.push((-1636));
	int int0 = stack0.pop();

Test Case#3	Test Case#4

Stack stack0 = new Stack();
try {
stack0.pop();
}
catch(EmptyStackException e)
{
}
	Stack stack0 = new Stack();
	try {
	stack0.top();
	}
	catch(EmptyStackException e)
	{
	}

Oregon State University

CDATAReader Class example

```
1.    class CDATAReader extends Reader {
2.        private IXMLReader reader;
3.        private char savedChar;
4.        private boolean atEndOfData;
5.    CDATAReader(IXMLReader reader){
6.        this.reader = reader;
7.        this.savedChar = 0;
8.        this.atEndOfData = false;
9.    }
10.   public int read(char[] buffer,int offset,int size)throws IOException
11.   { …
12.      while (…) {
13.          Char ch =this.savedChar;
14.          if (ch == 0)
15.              ch = this.reader.read();//B1
16.          else
17.              this.savedChar = 0; //B2
18.          if (ch == ']') {
19.               char ch2 = this.reader.read(); //B3
20.               if (ch2 == ']')
21.               … more if statements …
22.           }
23.        }…
24.   }
25.   … 3 more methods …
26.   }
27.   public class StdXMLReader implements IXMLReader{
28.    …
29.    public static IXMLReader stringReader(String str){
30.         return new StdXMLReader(new StringReader(str));
31.    }
32.     … 20 more methods …
33.   }
```

http://nanoxml.sourceforge.net/orig/

The class **CDATAReader** from **nanoxml** and contains only **4 public methods**

The **CDATAReader** and **StdXMLReader** objects must be in desired states to cover target branches

This method returns a stdXMLReader object, which can be used as a parameter

Oregon State University

CDATAReader Class example

```
1.    class CDATAReader extends Reader {
2.        private IXMLReader reader;
3.        private char savedChar;
4.        private boolean atEndOfData;
5.    CDATAReader(IXMLReader reader){
6.        this.reader = reader;
7.        this.savedChar = 0;
8.        this.atEndOfData = false;
9.    }
10.   public int read(char[] buffer,int offset,int size)throws IOException
11.   { …
12.      while (…) {
13.          Char ch =this.savedChar;
14.          if (ch == 0)
15.              ch = this.reader.read();//B1
16.          else
17.              this.savedChar = 0; //B2
18.          if (ch == ']') {
19.               char ch2 = this.reader.read()
20.               if (ch2 == ']')
21.               … more if statements …
22.           }
23.        }…
24.   }
25.   … 3 more methods …
26.   }
27.   public class StdXMLReader implements IXMLReader{
28.    …
29.    public static IXMLReader stringReader(String str){
30.         return new StdXMLReader(new StringReader(str));
31.    }
32.     … 20 more methods …
33.   }
```

http://nanoxml.sourceforge.net/orig/

The class **CDATAReader** from **nanoxml** and contains only **4 public methods**

Test Case#1

```
StdXMLReader stdXMLReader0 =
(StdXMLReader)StdXMLReader.stringReader("M2{Ivt=OB$")
;
CDATAReader cDATAReader0 = new
CDATAReader(stdXMLReader0);
    char[] charArray0 = new char[23];
int int0 = cDATAReader0.read(charArray0, (int)1, (int) 5);
```

Oregon State University

References:
Young, Michal, and Mauro Pezze. "Software Testing and Analysis: Process, Principles and Techniques." (2005).Chapter 15
Booch, Grady. Object oriented analysis & design with application. Pearson Education India, 2006.
https://www.st.cs.uni-saarland.de/edu
https://courses.cs.ut.ee/MTAT.03.159/
Goffi, Alberto, et al. "Search-based synthesis of equivalent method sequences." Proceedings of the 22nd ACM SIGSOFT International Symposium on Foundations of Software Engineering. ACM, 2014.