Peer Assisted Study Session

FIT2099 - Week 7

Monash University

Objectives

- Practice using exceptions
- Practice using generics

Estimated Time

Question 1 - Question 2 (10 Minutes)
Question 3 (10 Minutes)
Question 4 (10 Minutes)
Question 5 (20 Minutes)

Questions

```
class Main {
 public static void f() throws CriticalException {
   throw new CriticalException();
 }
  public static void cleanup() throws AvgException {
   throw new AvgException();
 }
  public static void main(String[] args) {
   try {
     try {
       f();

     } finally {
       cleanup();
     }
   } catch (Exception e) {
     System.out.println(e);
   }
 }
}
```

```
class CriticalException extends Exception {
 public String toString() {
   return "CriticalException";
 }
}

class AvgException extends Exception {
 public String toString() {
   return "AvgException";
 }
}
```

1. Explain what is wrong with the above code in the way it handles exceptions [hint: look at the finally clause]

The lost exception - Critical Exception not printed

2. Explain how to fix it and make changes to the code to show how you would fix it

```
public static void main(String[] args) {
   try {
     try {
       f();
     } catch (Exception e) {
       System.out.println(e);
     } finally {
       cleanup();
     }
   } catch (Exception e) {
     System.out.println(e);
   }
 }
```

3. Write a 2-tuple class with the methods: toString, getFirst, getSecond. Define a constructor for the tuple that accepts the pair of objects that the tuple will hold. Your tuple should work for any pair of types.

```
class Tuple<A, B> {
 private A first;
 private B second;
  public Tuple(A first, B second) {
   this.first = first;
   this.second = second;
 }
  public A getFirst() { return first; }
 public B getSecond() { return second; }
  public String toString() {
   return "(" + first + ", " + second + ")";
 }
}
```

4. Extend the tuple class to create a 2D vector class. Define a method add that returns a new 2D vector by adding the vector to the argument vector. [hint: extend Tuple<Double, Double>; for extra challenge, make your vector work for any numeric type (int, double, float, etc.)]

```
class Main {

public static void main(String[] args) {
  Vector2d vec1 = new Vector2d(2.0,5.0);
  Vector2d vec2 = new Vector2d(3.0,1.0);
  System.out.println(vec1.add(vec2));
}
}
```

```
class Tuple<A, B> {
private A first;
private B second;
 public Tuple(A first, B second) {
  this.first = first;
```

```
  this.second = second;
}
 public A getFirst() { return first; }
public B getSecond() { return second; }
 public String toString() {
  return "(" + first + ", " + second + ")";
}
}

class Vector2d extends Tuple<Double, Double> {
public Vector2d(Double x, Double y) {
  super(x, y);
}
 public Vector2d add(Vector2d other) {
  return new Vector2d(
      getFirst() + other.getFirst(),
      getSecond() + other.getSecond()
    );
}
}
```

5. Implement a generic stack class that works for any type of stack (e.g. stack of integers, stack of strings, etc) and that supports push and pop. Decide how to handle failure on either operation.
   [hint: use an ArrayList as the container inside your stack; use a plain array if you want an extra challenge]

```
import java.util.*;

class Main {
 public static void main(String[] args) {
   Stack<Integer> s = new Stack<>();
   s.push(1);
   s.push(2);
   System.out.println(s);
   int x = s.pop();
   assert x == 2;
```

```
    System.out.println(s);
  }
}

class StackEmptyError extends RuntimeException {
}

class Stack<A> {
 private ArrayList<A> stack = new ArrayList<>();
 private int top = -1;
  public A pop() {
    if (top == -1)
      throw new StackEmptyError();

    A result = stack.get(top);
    stack.remove(top);
    top--;
    return result;
  }
  public void push(A a) {
    stack.add(a);
    top++;
  }
  public String toString() {
    String s = "[ ";
    for (A a : stack)
      s = s + a + " ";
    return s + "]";
  }
}
```