Peer Assisted Study Session

FIT2099 - Week 6

Monash University

Objectives

- Understand an inner class
- How interfaces can be used
- Understand Error handling basics

Estimated Time

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

Questions

1. What is an inner class? Explain

It's possible to place a class definition within another class definition. This is known as an Inner Class.

2. Write a class named Outer that contains an inner class named Inner. Add a method to Outer that returns an object of type Inner. In main(), create and initialize a reference to an Inner.

```
class Outer {
 class Inner {
   { System.out.println("Inner created."); }
 }

 Inner getInner() {
   return new Inner();
 }
}

class Main {
```

```
 public static void main(String[] args) {
   Outer o = new Outer();
   Outer.Inner i = o.getInner();
 }
}
```

3. Separation of interface from implementation allows more reusable code. As an example, consider the Scanner class defined in java.util. It can be used as a simple parser for whitespace separated tokens from any character source. One of its constructors accepts a java.lang.Readable as a source of characters. This means that the scanner can work with a source that we define ourselves, not just files on secondary storage. Given this information, define a class RandomWords that implements Readable and acts as a finite source of randomly constructed words of some arbitrary length (not necessarily dictionary words) for parsing by Scanner. Then use the Scanner to extract 5 words from our newly defined source and print them to console.

   [Hint: refer to the javadocs for java.lang.Readable, java.util.Scanner, java.util.Random]

```
import java.util.*;
import java.nio.*;

class Main {
public static void main(String[] args) {
  Scanner s = new Scanner(new RandomWords());
  while (s.hasNext())
    System.out.println(s.next());
}
}

class RandomWords implements Readable {
  public final char[] letters =
  "abcdefghijklmnopqrstuvwxyz".toCharArray();
 public Random rand = new Random();
 public final int wordLength = 9;
 public int wordCount = 5;

 public int read(CharBuffer cb) {
```

```
            // check if no more words
            if (wordCount == 0)
              return -1;

            // generate a word
            for (int i = 0; i < wordLength; i++){
                  cb.append(letters[ rand.nextInt(letters.length) ]);
            }
            cb.append(" ");
            wordCount--;
            return wordLength + 1;
      }
      }
```

4. Create a class with a main() that throws an object of class Exception inside the try block. Give the constructor for Exception a String argument. Catch the exception inside a catch clause and print the String argument. Add a finally clause and print a message to prove you were there.

```
class Main{
 public static void main(String args[]){
   try{
     throw new Exception("An exception in main");
   }catch(Exception e){
     System.out.println("e.getMessage() = " + e.getMessage());
   }finally{
     System.out.println("In finally clause");
   }
 }
}
```

5. What is checked and unchecked exception? State two differences

Checked: are the exceptions that are checked at compile time.
FileNotFoundException.
IOException

If a client can reasonably be expected to recover from an exception, make it a checked exception.
Eg FileNotFound can ask to enter again

Unchecked are the exceptions that are not checked at compile time.
Error
RuntimeException
NullPointerException

If a client cannot do anything to recover from the exception, make it an **unchecked exception**
Eg NullPointerException, Network Error

Error and RuntimeException classes are unchecked exceptions, everything else under throwable is checked.