MIT World Peace University

Object Oriented Programming with Java and C++
Second Year B. Tech, Semester 1

Theory Assignment

Assignment No. 2

Prepared By

Krishnaraj Thadesar
Cyber Security and Forensics
Batch A2, PA 20

November 28, 2022

1 Questions

1.1 What is the difference between checked and unchecked exceptions in Java? Explain with suitable example.

1.1.1 Checked Exceptions

In broad terms, a checked exception (also called a logical exception) in Java is something that has gone wrong in your code and is potentially recoverable. For example, if there's a client error when calling another API, we could retry from that exception and see if the API is back up and running the second time. A checked exception is caught at compile time so if something throws a checked exception the compiler will enforce that you handle it.

Example:

```java
import java.io.File;
import java.io.FileInputStream;

public class CheckedException {
  public void readFile() {
    String fileName = "file does not exist";
    File file = new File(fileName);
    FileInputStream stream = new FileInputStream(file);
  }
}
```

You simply wrap the Java code which throws the checked exception within a try catch block. This now allows you to process and deal with the exception. With this approach it's very easy to swallow the exception and then carry on like nothing happened. Later in the code when what the method was doing is required you may find yourself with our good friend the NullPointerException.

1.1.2 Unchecked Exceptions

An unchecked exception (also known as an runtime exception) in Java is something that has gone wrong with the program and is unrecoverable. Just because this is not a compile time exception, meaning you do not need to handle it, that does not mean you don't need to be concerned about it.

The most common Java unchecked exception is the good old NullPointerException which is when you are trying to access a variable or object that doesn't exist.

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

public class IndexOutOfBounds {
  public static void main(String[] args) {
    List<String> lst = new ArrayList<>();
    lst.add("item-1");
    lst.add("item-2");
    lst.add("item-3");
    var result = lst.get(lst.size());
  }
}
```

1.1.3 Differncees

- A checked exception is caught at compile time whereas a runtime or unchecked exception is, as it states, at runtime.
- A checked exception must be handled either by re-throwing or with a try catch block, whereas an unchecked isn't required to be handled.
- A runtime exception is a programming error and is fatal whereas a checked exception is an exception condition within your code's logic and can be recovered or re-tried from.

1.2 What is a stream? Explain the different file stream classes with its use?

1.2.1 Stream

A stream is a sequence of data. It is a sequence of bytes. It is a sequence of characters. It is a sequence of objects. It is a sequence of anything. A stream is a sequence of data. It is a sequence of bytes. It is a sequence of characters. It is a sequence of objects. It is a sequence of anything.

In general, a Stream will be an input stream or, an output stream.

- InputStream - This is used to read data from a source.
- OutputStream - This is used to write data to a destination.

1.2.2 FileInputStream

This stream is used for reading data from the files. Objects can be created using the keyword new and there are several types of constructors available.

Once you have InputStream object in hand, then there is a list of helper methods which can be used to read to stream or to do other operations on the stream.

- **public void close()** throws IOException: This method closes the file output stream. Releases any system resources associated with the file. Throws an IOException.
- **protected void finalize()**throws IOException : This method cleans up the connection to the file. Ensures that the close method of this file output stream is called when there are no more references to this stream. Throws an IOException.
- **public int read(int r)throws IOException:** This method reads the specified byte of data from the InputStream. Returns an int. Returns the next byte of data and -1 will be returned if it's the end of the file.
- **public int read(byte[] r) throws IOException:** This method reads r.length bytes from the input stream into an array. Returns the total number of bytes read. If it is the end of the file, -1 will be returned.
- **public int available() throws IOException:** Gives the number of bytes that can be read from this file input stream. Returns an int.

1.2.3 FileOutputStream

FileOutputStream is used to create a file and write data into it. The stream would create a file, if it doesn't already exist, before opening it for output.

Here are two constructors which can be used to create a FileOutputStream object. Once you have OutputStream object in hand, then there is a list of helper methods, which can be used to write to stream or to do other operations on the stream.

- **public void close()** throws IOException: This method closes the file output stream. Releases any system resources associated with the file. Throws an IOException.
- **protected void finalize()throws IOException :** This method cleans up the connection to the file. Ensures that the close method of this file output stream is called when there are no more references to this stream. Throws an IOException.
- **public void write(int w)throws IOException**: This methods writes the specified byte to the output stream.
- **public void write(byte[] w):** Writes w.length bytes from the mentioned byte array to the OutputStream.

Example

```java
import java.io.*;
public class fileStreamTest {

  public static void main(String args[]) {

    try {
      byte bWrite [] = {11,21,3,40,5};
      OutputStream os = new FileOutputStream("test.txt");
      for(int x = 0; x < bWrite.length ; x++) {
      os.write( bWrite[x] );   // writes the bytes
      }
      os.close();

      InputStream is = new FileInputStream("test.txt");
      int size = is.available();

      for(int i = 0; i < size; i++) {
      System.out.print((char)is.read() + "  ");
      }
      is.close();
    } catch (IOException e) {
      System.out.print("Exception");
    }
  }
}
```

1.3 What is the difference between the paint() and repaint() methods? Explain with suitable examples.

1.3.1 paint()

The paint() method is called by the AWT when the component needs to be painted. It is called automatically when the component is first shown on the screen. It is also called whenever the contents of the component need to be updated.

Syntax:

```java
public void paint(Graphics g)
```

Parameters: g - the specified Graphics context
Example:

```java

import java.awt.*;
public class paint extends Frame {

  public void paint(Graphics g) {
    g.drawString("Welcome", 50, 50);
    g.drawLine(20, 30, 20, 300);
    g.drawRect(70, 100, 30, 30);
    g.fillRect(170, 100, 30, 30);
    g.drawOval(70, 200, 30, 30);
  }

  public static void main(String args[]) {
    paint p = new paint();
    p.setSize(400, 400);
    p.setVisible(true);
  }
}
```

1.3.2 repaint()

The repaint() method is used to force a call to the paint() method. It is used to redraw the component. It is used to update the component.

Syntax:

```java
public void repaint()
```

Example:

```java
import java.awt.*;
public class repaint extends Frame {

  public void paint(Graphics g) {
    g.drawString("Welcome", 50, 50);
    g.drawLine(20, 30, 20, 300);
    g.drawRect(70, 100, 30, 30);
    g.fillRect(170, 100, 30, 30);
    g.drawOval(70, 200, 30, 30);
  }

  public static void main(String args[]) {
    repaint p = new repaint();
    p.setSize(400, 400);
    p.setVisible(true);
    p.repaint();
  }
}
```

1.4 Write the Java Swing Code that creates a label displaing the text as "Welcome to MITWPU, Pune", with the italics and font size as 15.

```java
import javax.swing.*;
import java.awt.event.*;
import java.awt.*;

class Main extends JFrame {
    Main() {
    JLabel lbl = new JLabel("Welcome to MITWPU, Pune");
    lbl.setFont(new Font ("Ariel", Font.ITALIC, 15));
    setDefaultCloseOperation(JFrame.EXIT_ON_CLOSE); // setting close operation.
    lbl.setBounds(50, 0, 500, 100);
    setLayout(null); // setting layout using FlowLayout object
    setSize(300, 150); // setting size of Jframe
    add(lbl); // adding Yes button to frame.
    setVisible(true);
  }


  public static void main(String[] args) {
    new Main();
  }
}
```