



LAB - 7

Java Programming Lab

Topics Covered

Exception Handling,
Applet

Syeda Reeha Quasar

14114802719

4C7

EXPERIMENT – 7.1

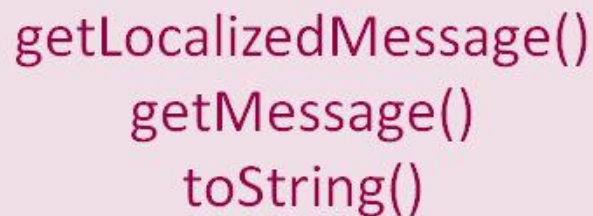
Aim:

Write an application that shows how to create a user-defined exception.

Theory:

User Defined Exception or custom exception is creating your own exception class and throws that exception using 'throw' keyword. This can be done by extending the class Exception.

Exception



```
getLocalizedMessage()  
getMessage()  
toString()
```

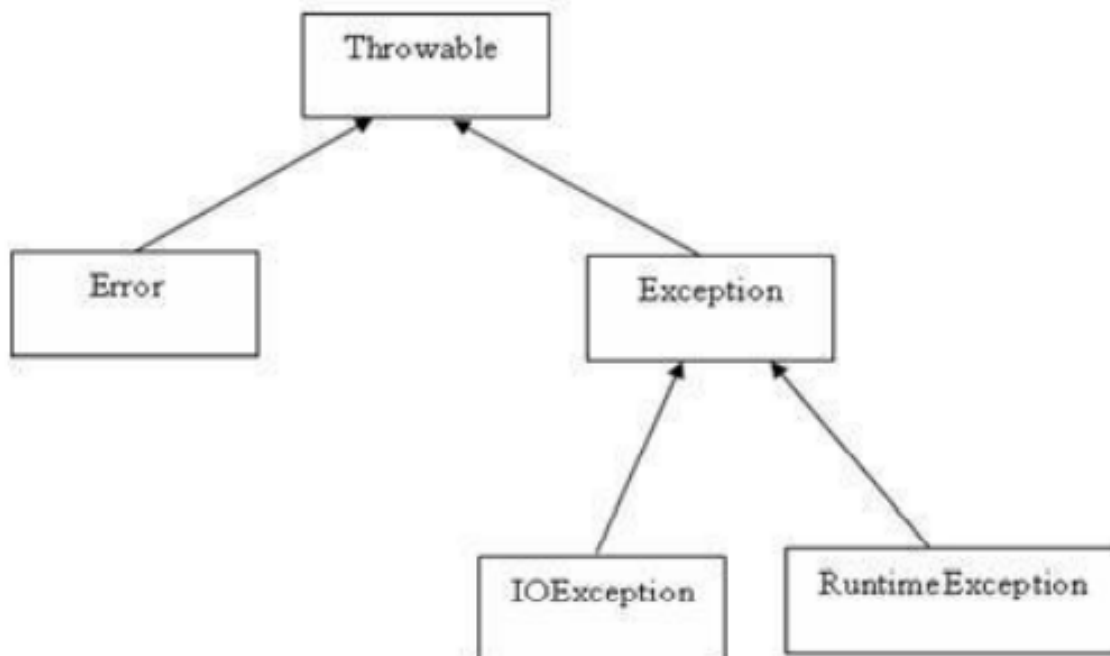
There is no need to override any of the above methods available in the Exception class, in your derived class. But practically, you will require some amount of customizing as per your programming needs.

An exception is a problem that arises during the execution of a program. An exception can occur for many different reasons, including the following:

- A user has entered invalid data.
- A file that needs to be opened cannot be found.
- A network connection has been lost in the middle of communications, or the JVM has run out of memory.
- Some of these exceptions are caused by user error, others by programmer error, and others by physical resources that have failed in some manner.

To understand how exception handling works in Java, you need to understand the three categories of exceptions:

- **Checked exceptions:** A checked exception is an exception that is typically a user error or a problem that cannot be foreseen by the programmer. For example, if a file is to be opened, but the file cannot be found, an exception occurs. These exceptions cannot simply be ignored at the time of compilation.
- **Runtime exceptions:** A runtime exception is an exception that occurs that probably could have been avoided by the programmer. As opposed to checked exceptions, runtime exceptions are ignored at the time of compilation.
- **Errors:** These are not exceptions at all, but problems that arise beyond the control of the user or the programmer. Errors are typically ignored in your code because you can rarely do anything about an error. For example, if a stack overflow occurs, an error will arise. They are also ignored at the time of compilation.



Concepts:

- **class** keyword is used to declare a class in Java.
- **public** keyword is an access modifier that represents visibility. It means it is visible to all.
- **static** is a keyword. If we declare any method as static, it is known as the static method. The core advantage of the static method is that there is no need to

create an object to invoke the static method. The main() method is executed by the JVM, so it doesn't require creating an object to invoke the main() method. So, it saves memory.

- **void** is the return type of the method. It means it doesn't return any value.
- **main** represents the starting point of the program.
- **String[] args** or **String args[]** is used for command line argument. We will discuss it in coming section.
- **System.out.println()** is used to print statement. Here, System is a class, out is an object of the PrintStream class, println() is a method of the PrintStream class. We will discuss the internal working of System.out.println() statement in the coming section.
- **Java Utils:** Resources Job Search Discussion. Java. util package contains the **collections framework, legacy collection classes, event model, date and time facilities, internationalization, and miscellaneous utility classes**. This reference will take you through simple and practical methods available in java.
- util. Java util package contains **collection framework, collection classes**, classes related to date and time, event model, internationalization, and miscellaneous utility classes. ... On importing this package, you can access all these classes and methods.
- **Scanner** is a class in **java. util package used for obtaining the input of** the primitive types like int, double, etc. and strings. ... To create an object of Scanner class, we usually pass the predefined object System.in, which represents the standard input stream.
- A switch statement **allows a variable to be tested for equality against a list of values**. Each value is called a case, and the variable being switched on is checked for each case.
- The input is **the data that we give to the program**. The output is the data what we receive from the program in the form of result. Stream represents flow of data or the sequence of data.

Source Code:

```
public class JavaException {
    public static void main(String args[]) {
        try {
            throw new MyException(2);
            // throw is used to create a new exception and throw it.
        } catch (MyException e) {
            System.out.println(e);
        }
    }
}

class MyException extends Exception {
    int a;

    MyException(int b) {
        a = b;
    }

    public String toString() {
        return ("Exception Number = " + a);
    }
}
```

Output:

```
PS E:\sem 5\java> javac .\JavaException.java
PS E:\sem 5\java> java JavaException
Exception Number = 2
```

```
Exception Number = 2
```

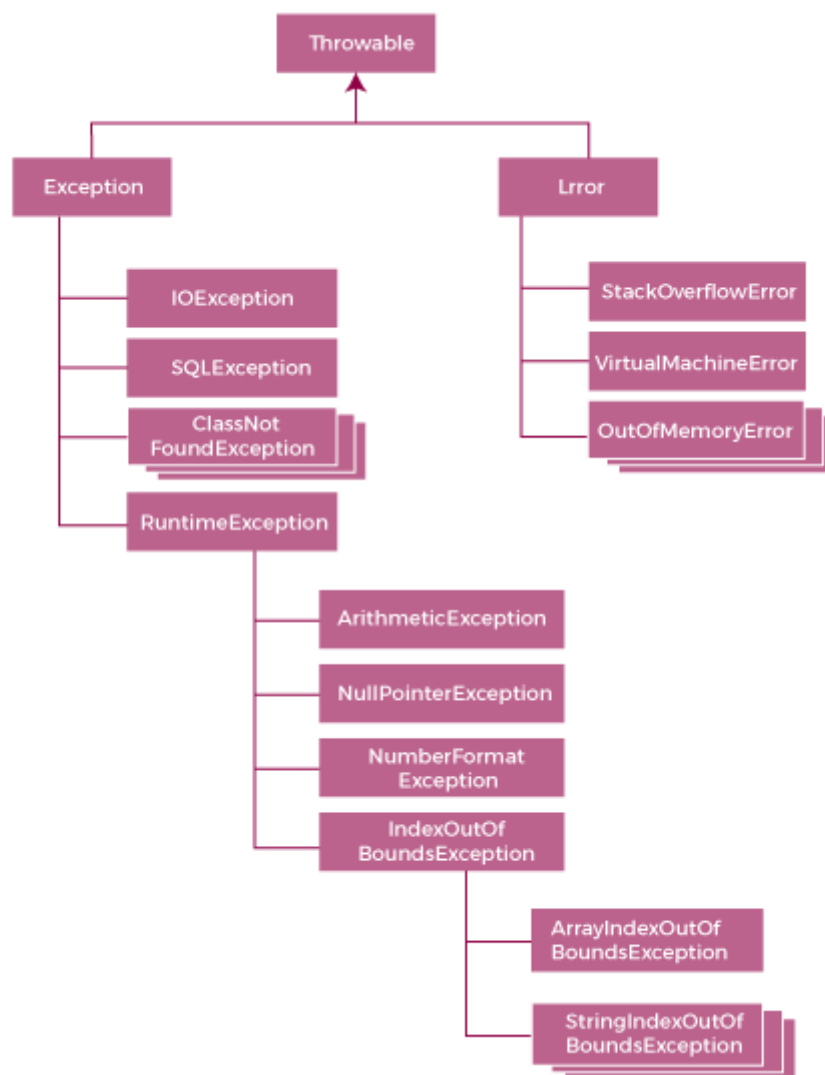
EXPERIMENT – 7.2

Aim:

Create a customized exception and also make use of all the 5 exception keywords.

Theory:

The **Exception Handling in Java** is one of the powerful *mechanism to handle the runtime errors* so that the normal flow of the application can be maintained.



The `java.lang.Throwable` class is the root class of Java Exception hierarchy inherited by two subclasses: `Exception` and `Error`.

Keyword	Description
try	The "try" keyword is used to specify a block where we should place an exception code. It means we can't use try block alone. The try block must be followed by either catch or finally.
catch	The "catch" block is used to handle the exception. It must be preceded by try block which means we can't use catch block alone. It can be followed by finally block later.
finally	The "finally" block is used to execute the necessary code of the program. It is executed whether an exception is handled or not.
throw	The "throw" keyword is used to throw an exception.
throws	The "throws" keyword is used to declare exceptions. It specifies that there may occur an exception in the method. It doesn't throw an exception. It is always used with method signature.

Source Code:

```
import java.io.IOException;

public class exceptions {
    void divideSt() {
        try {
            throw new ArithmeticException("func Can't divide a number by 0 --
-> this is by throws");
        } catch (Exception e) {
            System.out.println("Exception handles of throw");
        }
    }

    public static void main(String args[]) throws ArithmeticException {
        exceptions obj = new exceptions();
        try {
```



```
        int data = 100 / 0;

        throw new ArithmeticException("Can't divide a number by 0 --->
this is by throws");
    } catch (ArithmeticException e) {
        System.out.println("Exception handles of throw2" + e);
        obj.divideSt();

    } catch (Exception e) {
        System.out.println("Exception handles of throw3" + e);

    } finally {
        System.out.println("finally executed");
        System.out.println("normal flow...");
    }
}
}
```

Output:

```
PS E:\sem 5\java> javac .\exceptions.java
PS E:\sem 5\java> java exceptions
Exception handles of throw2java.lang.ArithmeticException: / by zero
Exception handles of throw
finally executed
normal flow...
```

```
Exception handles of throw2java.lang.ArithmeticException: / by zero
Exception handles of throw
finally executed
normal flow...
```

EXPERIMENT – 7.3

Aim:

Write an application that creates an 'interface' and implements it.

Theory:

All applets are subclasses of Applet. Thus, all applets must import `java.applet`. Applets must also import `java.awt`. Recall that AWT stands for the Abstract Window Toolkit. Since all applets run in a window, it is necessary to include support for that window. Applets are not executed by the console-based Java run-time interpreter. Rather, they are executed by either a Web browser or an applet viewer. The figures shown in this chapter were created with the standard applet viewer, called applet viewer, provided by the SDK. But you can use any applet viewer or browser you like. Execution of an applet does not begin at `main()`. Actually, few applets even have `main()` methods. Instead, execution of an applet is started and controlled with an entirely different mechanism, which will be explained shortly. Output to your applet's window is not performed by `System.out.println()`. Rather, it is handled with various AWT methods, such as `drawString()`, which outputs a string to a specified X,Y location. Input is also handled differently than in an application. Once an applet has been compiled, it is included in an HTML file using the `APPLET` tag. The applet will be executed by a Java-enabled web browser when it encounters the `APPLET` tag within the HTML file. To view and test an applet more conveniently, simply include a comment at the head of your Java source code file that contains the `APPLET` tag. This way, your code is documented with the necessary HTML statements needed by your applet, and you can test the compiled applet by starting the applet viewer with your Java source code file specified as the target.

Applet An applet is a Java program that runs in a Web browser. An applet can be a fully functional Java application because it has the entire Java API at its disposal. There are some important differences between an applet and a standalone Java application, including the following:

- An applet is a Java class that extends the `java.applet.Applet` class.
- A `main()` method is not invoked on an applet, and an applet class will not define `main()`.
- Applets are designed to be embedded within an HTML page.
- When a user views an HTML page that contains an applet, the code for the applet is downloaded to the user's machine.

- A JVM is required to view an applet. The JVM can be either a plug-in of the Web browser or a separate runtime environment.
- The JVM on the user's machine creates an instance of the applet class and invokes various methods during the applet's lifetime.
- Applets have strict security rules that are enforced by the Web browser. The security of an applet is often referred to as sandbox security, comparing the applet to a child playing in a sandbox with various rules that must be followed.

Other classes that the applet needs can be downloaded in a single Java Archive (JAR) file.

Life Cycle of an Applet:

Four methods in the Applet class give you the framework on which you build any serious applet:

- **init:** This method is intended for whatever initialization is needed for your applet. It is called after the param tags inside the applet tag have been processed.
- **start:** This method is automatically called after the browser calls the init method. It is also called whenever the user returns to the page containing the applet after having gone off to other pages.
- **stop:** This method is automatically called when the user moves off the page on which the applet sits. It can, therefore, be called repeatedly in the same applet.
- **destroy:** This method is only called when the browser shuts down normally. Because applets are meant to live on an HTML page, you should not normally leave resources behind after a user leaves the page that contains the applet.
- **paint:** Invoked immediately after the start() method, and also any time the applet needs to repaint itself in the browser. The paint() method is actually inherited from the java.awt.

Application Conversion to Applets: It is easy to convert a graphical Java application (that is, an application that uses the AWT and that you can start with the java program launcher) into an applet that you can embed in a web page. Here are the specific steps for converting an application to an applet.

- Make an HTML page with the appropriate tag to load the applet code.

- Supply a subclass of the JApplet class. Make this class public. Otherwise, the applet cannot be loaded.
- Eliminate the main method in the application. Do not construct a frame window for the application. Your application will be displayed inside the browser.
- Move any initialization code from the frame window constructor to the init method of the applet. You don't need to explicitly construct the applet object. the browser instantiates it for you and calls the init method.
- Remove the call to setSize; for applets, sizing is done with the width and height parameters in the HTML file.
- Remove the call to setDefaultCloseOperation. An applet cannot be closed; it terminates when the browser exits.
- If the application calls setTitle, eliminate the call to the method. Applets cannot have title bars. (You can, of course, title the web page itself, using the HTML title tag.)
- Don't call setVisible(true). The applet is displayed automatically.

Source Code:

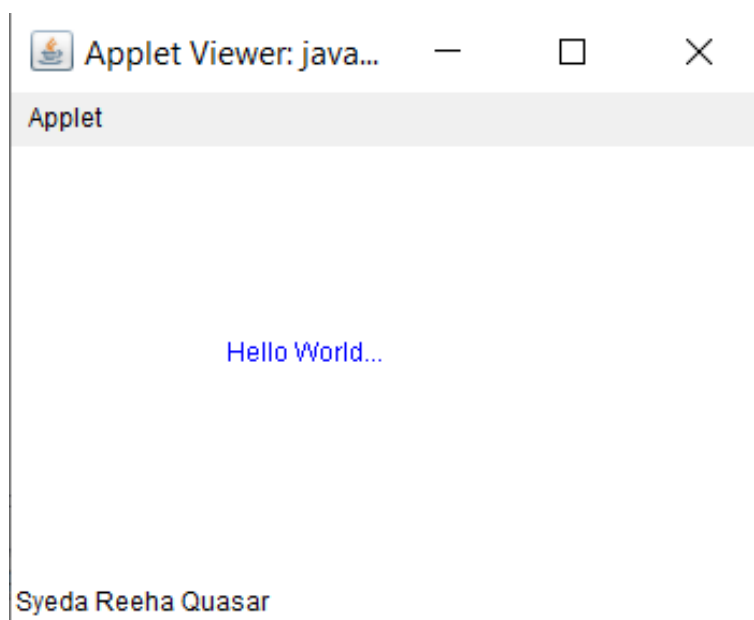
```
package javaapplication1;

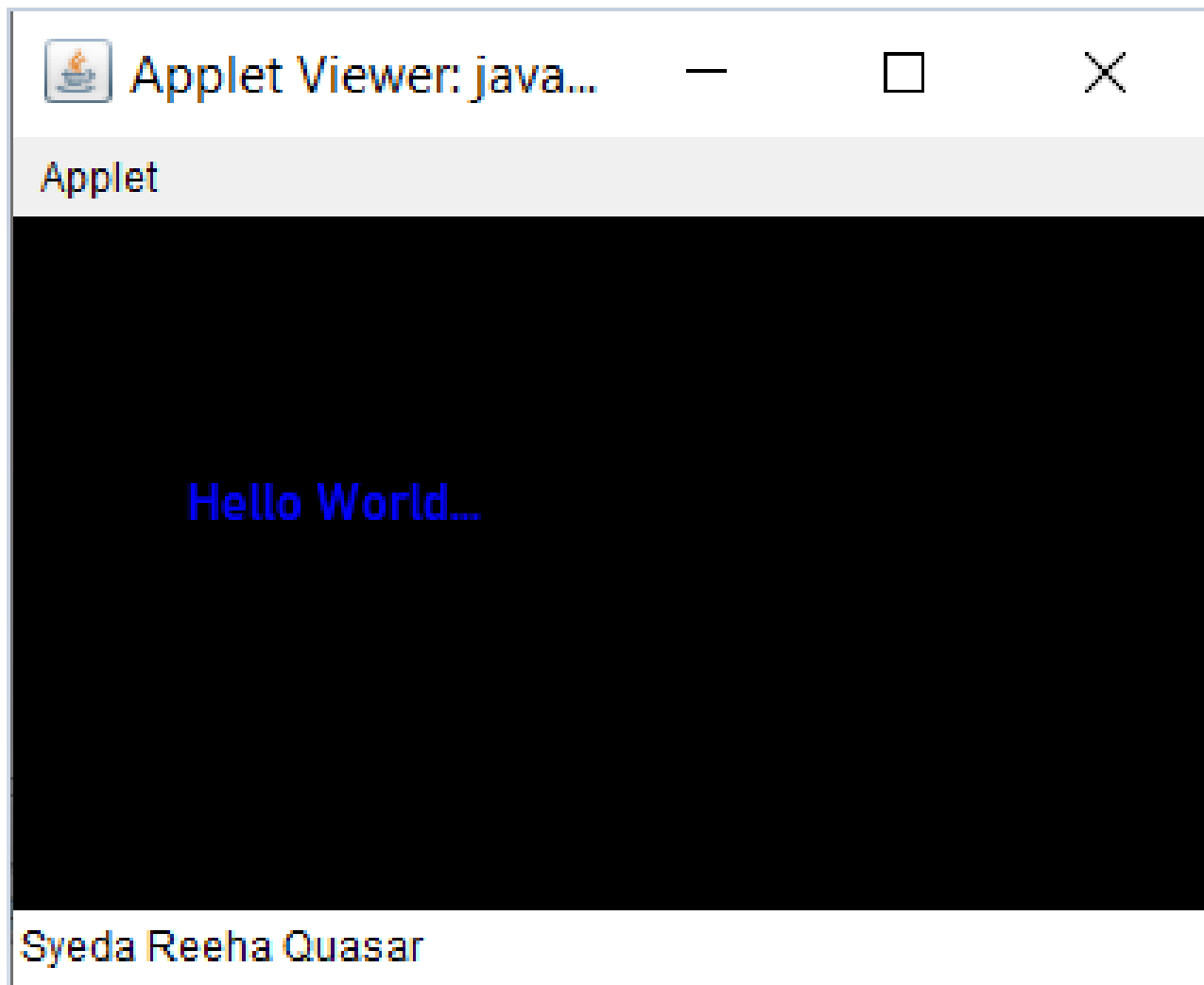
import javax.swing.JApplet;
import java.awt.*;

/**
 *
 * @author reeha
 */
public class NewJApplet extends JApplet {
```

```
public void init() {  
    getContentPane().setBackground(Color.BLACK);  
}  
  
public void paint(Graphics g) {  
    // set color to blue  
    g.setColor(Color.blue);  
  
    // print hello world  
    g.drawString("Hello World...", 100, 100);  
  
    showStatus("Syeda Reeha Quasar");  
}  
}
```

Output:





EXPERIMENT – 7.4

Aim:

Develop an analog clock using applet.

Theory:

Four methods in the Applet class give you the framework on which you build any serious applet:

- **init:** This method is intended for whatever initialization is needed for your applet. It is called after the param tags inside the applet tag have been processed.
- **start:** This method is automatically called after the browser calls the init method. It is also called whenever the user returns to the page containing the applet after having gone off to other pages.
- **stop:** This method is automatically called when the user moves off the page on which the applet sits. It can, therefore, be called repeatedly in the same applet.
- **destroy:** This method is only called when the browser shuts down normally. Because applets are meant to live on an HTML page, you should not normally leave resources behind after a user leaves the page that contains the applet.
- **paint:** Invoked immediately after the start() method, and also any time the applet needs to repaint itself in the browser. The paint() method is actually inherited from the java.awt.

Application Conversion to Applets: It is easy to convert a graphical Java application (that is, an application that uses the AWT and that you can start with the java program launcher) into an applet that you can embed in a web page. Here are the specific steps for converting an application to an applet.

- Make an HTML page with the appropriate tag to load the applet code.
- Supply a subclass of the JApplet class. Make this class public. Otherwise, the applet cannot be loaded.

- Eliminate the main method in the application. Do not construct a frame window for the application. Your application will be displayed inside the browser.
- Move any initialization code from the frame window constructor to the init method of the applet. You don't need to explicitly construct the applet object. the browser instantiates it for you and calls the init method.
- Remove the call to setSize; for applets, sizing is done with the width and height parameters in the HTML file.
- Remove the call to setDefaultCloseOperation. An applet cannot be closed; it terminates when the browser exits.
- If the application calls setTitle, eliminate the call to the method. Applets cannot have title bars. (You can, of course, title the web page itself, using the HTML title tag.)
- Don't call setVisible(true). The applet is displayed automatically.

The Applet CLASS:

Every applet is an extension of the java.applet.Applet class. The base Applet class provides methods that a derived Applet class may call to obtain information and services from the browser context. These include methods that do the following:

- Get applet parameters
- Get the network location of the HTML file that contains the applet
- Get the network location of the applet class directory
- Print a status message in the browser
- Fetch an image
- Fetch an audio clip
- Play an audio clip
- Resize the applet

Additionally, the Applet class provides an interface by which the viewer or browser obtains information about the applet and controls the applet's execution.

The viewer may:

request information about the author, version and copyright of the applet

request a description of the parameters the applet recognizes

- initialize the applet
- destroy the applet
- start the applet's execution
- stop the applet's execution

The Applet class provides default implementations of each of these methods. Those implementations may be overridden as necessary.

Source Code:

```
// Java program to illustrate
// analog clock using Applets

import java.applet.Applet;
import java.awt.*;
import java.util.*;
import java.text.DateFormat;
import java.text.SimpleDateFormat;
import java.util.Date;
import java.util.Calendar;

public class clock extends Applet {

    @Override
    public void init()
    {
        // Applet window size & color
        this.setSize(new Dimension(800, 400));
        setBackground(new Color(50, 50, 50));
        new Thread() {
            @Override
            public void run()
            {
                while (true) {
                    repaint();
                    delayAnimation();
                }
            }
        }.start();
    }
}
```

```
}

// Animating the applet
private void delayAnimation()
{
    try {

        // Animation delay is 1000 milliseconds
        Thread.sleep(1000);
    }
    catch (InterruptedException e) {
        e.printStackTrace();
    }
}

// Paint the applet
@Override
public void paint(Graphics g)
{
    // Get the system time
    Calendar time = Calendar.getInstance();

    int hour = time.get(Calendar.HOUR_OF_DAY);
    int minute = time.get(Calendar.MINUTE);
    int second = time.get(Calendar.SECOND);
    Date date = time.getTime();
    DateFormat dateFormat = new SimpleDateFormat("yyyy-mm-dd
hh:mm:ss");
    String clkString = dateFormat.format(date);

    // 12 hour format
    if (hour > 12) {
        hour -= 12;
    }

    // Draw clock body center at (400, 200)
    g.setColor(Color.white);
    g.fillOval(300, 100, 200, 200);

    // Labeling
    g.setColor(Color.black);
    g.drawString("12", 390, 120);
    g.drawString("9", 310, 200);
    g.drawString("6", 400, 290);
    g.drawString("3", 480, 200);
```

```
// Declaring variables to be used
double angle;
int x, y;

// Second hand's angle in Radian
angle = Math.toRadians((15 - second) * 6);

// Position of the second hand
// with length 100 unit
x = (int)(Math.cos(angle) * 100);
y = (int)(Math.sin(angle) * 100);

// Red color second hand
g.setColor(Color.red);
g.drawLine(400, 200, 400 + x, 200 - y);

// Minute hand's angle in Radian
angle = Math.toRadians((15 - minute) * 6);

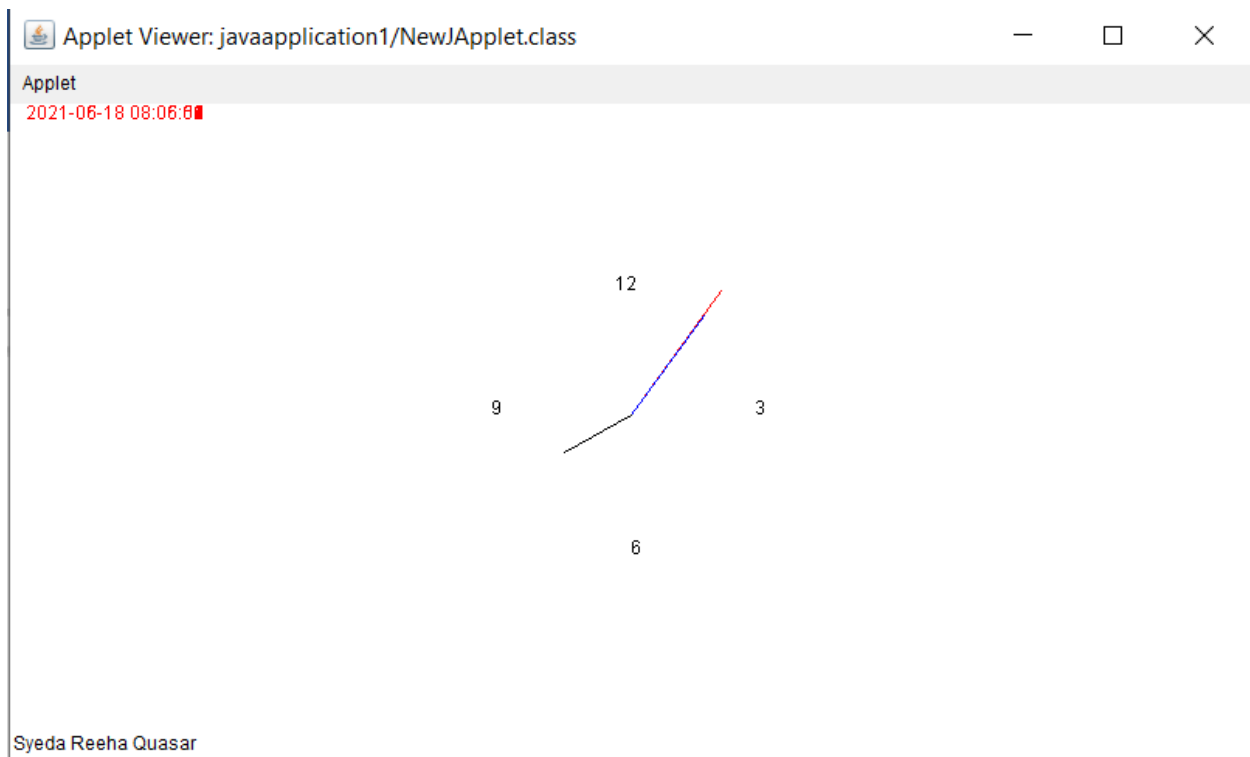
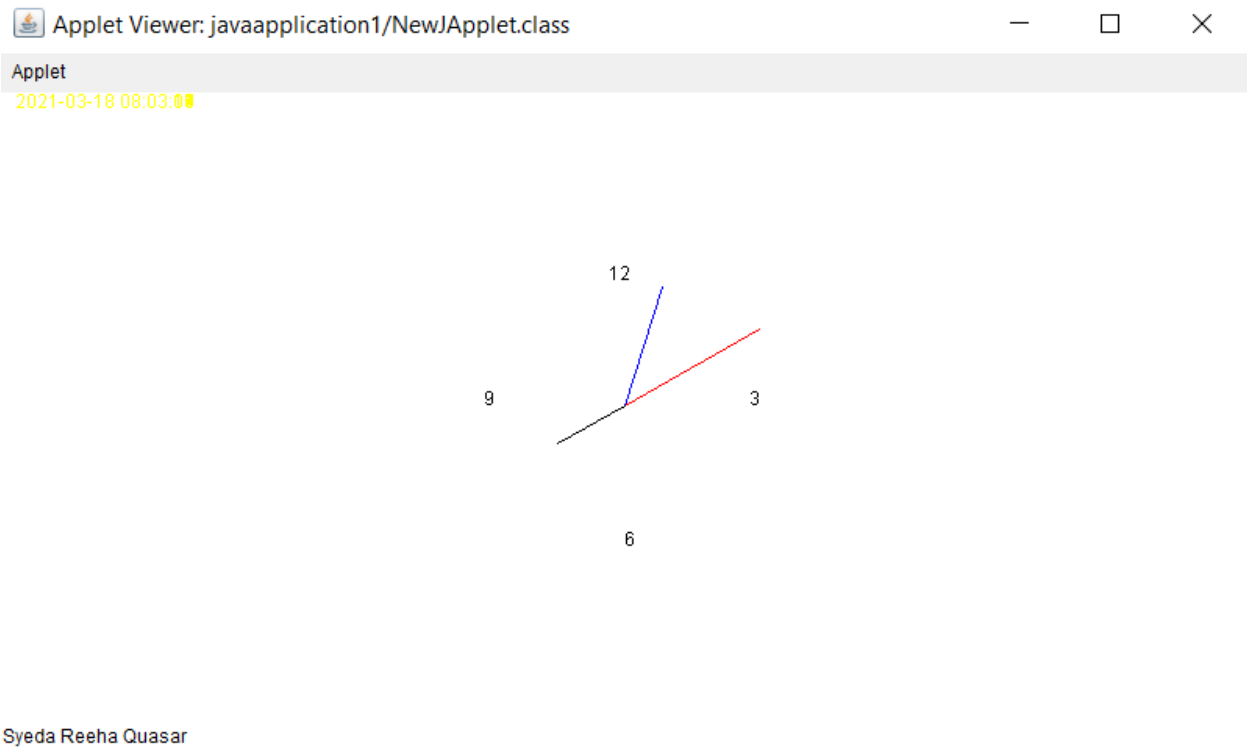
// Position of the minute hand
// with length 80 unit
x = (int)(Math.cos(angle) * 80);
y = (int)(Math.sin(angle) * 80);

// blue color Minute hand
g.setColor(Color.blue);
g.drawLine(400, 200, 400 + x, 200 - y);

// Hour hand's angle in Radian
angle = Math.toRadians((15 - (hour * 5)) * 6);

// Position of the hour hand
// with length 50 unit
x = (int)(Math.cos(angle) * 50);
y = (int)(Math.sin(angle) * 50);

// Black color hour hand
g.setColor(Color.black);
g.drawLine(400, 200, 400 + x, 200 - y);
    g.setColor(Color.yellow);
    g.drawString(clkString, 10, 10);
    showStatus("Syeda Reeha Quasar");
}
}
```

Output:

Viva Questions

1. What is the impact of declaring a method as final?

Ans.

A method declared as final can't be overridden. A sub-class can't have the same method signature with a different implementation.

2. How is final different from finally and finalize()?

Ans.

final is a modifier which can be applied to a class or a method or a variable. final class can't be inherited, final method can't be overridden and final variable can't be changed.

finally is an exception handling code section which gets executed whether an exception is raised or not by the try block code segment.

finalize() is a method of Object class which will be executed by the JVM just before garbage collecting object to give a final chance for resource releasing activity.

3. How will you communicate between two Applets?

Ans.

The simplest method is to use the static variables of a shared class since there's only one instance of the class and hence only one copy of its static variables. A slightly more reliable method relies on the fact that all the applets on a given page share the same AppletContext. We obtain this applet context as follows:

```
AppletContext ac = getAppletContext();
```

AppletContext provides applets with methods such as getApplet(name), getApplets(), getAudioClip, getImage, showDocument and showStatus().

4. Which classes can an applet extend?

Ans.

An applet can extend the `java.applet.Applet` class or the `java.swing.JApplet` class. The `java.applet.Applet` class extends the `java.awt.Panel` class and enables you to use the GUI tools in the AWT package. The `java.swing.JApplet` class is a subclass of `java.applet.Applet` that also enables you to use the Swing GUI tools.

5. How do you cause the applet GUI in the browser to be refreshed when data in it may have changed?

Ans.

You invoke the `repaint()` method. This method causes the `paint` method, which is required in any applet, to be invoked again, updating the display.