Inheritance

Sample Code

Below given is an example demonstrating Java inheritance. In this example you can observe two classes namely Calculation and My_Calculation. Using extends keyword the My_Calculation inherits the methods addition and Subtraction of Calculation class. Copy and paste the program given below in a file with name My_Calculation.java

```java
class Calculation{

    int z;

    public void addition(int x, int y){

    z=x+y;

    System.out.println("The sum of the given numbers:"+z);

    }

    public void Substraction(int x,int y){

    z=x-y;

    System.out.println("The difference between the given numbers:"+z);

    }

    }

    public class My_Calculation extends Calculation{

    public void multiplication(int x, int y){

    z=x*y;

    System.out.println("The product of the given numbers:"+z);

    }

    public static void main(String args[]){

    int a=20, b=10;

    My_Calculation demo = new My_Calculation();

    demo.addition(a, b);

    demo.Substraction(a, b);

    demo.multiplication(a, b);

    }
```

```
    }
```

Output:

```
The difference between the given numbers:10
The product of the given numbers:200
PS C:\Users\madet\OneDrive\Desktop\javaPrograms> ^C
PS C:\Users\madet\OneDrive\Desktop\javaPrograms>
PS C:\Users\madet\OneDrive\Desktop\javaPrograms>  c:; cd 'c:\Users\madet\OneDrive\Desktop\jav
aPrograms'; & 'C:\Program Files\Java\jre1.8.0_421\bin\java.exe' '-cp' 'C:\Users\madet\AppData
\Roaming\Code\User\workspaceStorage\84266f0d6e3a3bcc849fe3e6dff03fb0\redhat.java\jdt_ws\javaP
rograms_292963c0\bin' 'My_Calculation'
The sum of the given numbers:30
The difference between the given numbers:10
The product of the given numbers:200
```

The super keyword

The super keyword is similar to this keyword following are the scenarios where the super keyword is used. It is used to differentiate the members of superclass from the members of subclass, if they have same names. It is used to invoke the superclass constructor from subclass. Differentiating the members If a class is inheriting the properties of another class. And if the members of the superclass have the names same as the sub class, to differentiate these variables we use super keyword as shown below. super.variable super.method();

Sample Code

This section provides you a program that demonstrates the usage of the super keyword. In the given program you have two classes namely Sub_class and Super_class, both have a method named display with different implementations, and a variable named num with different values. We are invoking display method of both classes and printing the value of the variable num of both classes, here you can observe that we have used super key word to differentiate the members of super class from sub class.

```
class Super_class{

    int num=20;

    //display method of superclass

    public void display(){

    System.out.println("This is the display method of superclass");

    }

    }

    public class Sub_class extends Super_class {
```

```java
int num=10;
//display method of sub class
public void display(){
System.out.println("This is the display method of subclass");
}
public void my_method(){
//Instantiating subclass
Sub_class sub=new Sub_class();
//Invoking the display() method of sub class
sub.display();
//Invoking the display() method of superclass
super.display();
//printing the value of variable num of subclass
System.out.println("value of the variable named num in sub class:"+ sub.num);
//printing the value of variable num of superclass
System.out.println("value of the variable named num in super class:"+ super.num);
}
public static void main(String args[]){
Sub_class obj = new Sub_class();
obj.my_method();
}
}
```

```
PS C:\Users\madet\OneDrive\Desktop\javaPrograms>  & 'C:\Program Files\Java\jre1.8.0_421\bin\j
ava.exe' '-agentlib:jdwp=transport=dt_socket,server=n,suspend=y,address=localhost:58568' '-cp
' 'C:\Users\madet\AppData\Roaming\Code\User\workspaceStorage\84266f0d6e3a3bcc849fe3e6dff03fb0
\redhat.java\jdt_ws\javaPrograms_292963c0\bin' 'Sub_class'
This is the display method of subclass
This is the display method of superclass
value of the variable named num in sub class:10
value of the variable named num in super class:20
PS C:\Users\madet\OneDrive\Desktop\javaPrograms>
```

Invoking Superclass constructor

If a class is inheriting the properties of another class, the subclass automatically acquires the default constructor of the super class. But if you want to call a parametrized constructor of the super class, you need to use the super keyword as shown below. super(values);

Sample Code

The program given in this section demonstrates how to use the super keyword to invoke the parametrized constructor of the superclass. This program contains a super class and a sub class, where the super class contains a parametrized constructor which accepts a string value, and we used the super keyword to invoke the parametrized constructor of the super class.

```java
class Superclass{

    int age;

    Superclass(int age){

    this.age=age;

    }

    public void getAge(){

    System.out.println("The value of the variable named age in super class is: " +age);

    }

    }

    public class Subclass extends Superclass {

    Subclass(int age){

    super(age);

    }

    public static void main(String argd[]){

    Subclass s= new Subclass(24);

    s.getAge();

    }

    }
```
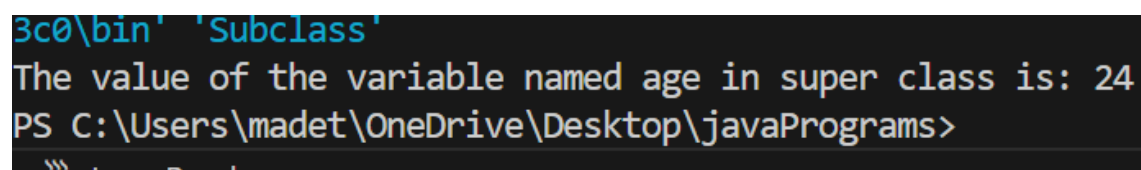
```
3c0\bin' 'Subclass'
The value of the variable named age in super class is: 24
PS C:\Users\madet\OneDrive\Desktop\javaPrograms>
```

Example

```java
class Animal{

}

class Mammal extends Animal{

}

class Reptile extends Animal{

}

public class Dog extends Mammal{

public static void main(String args[]){

Animal a = new Animal();

Mammal m = new Mammal();

Dog d = new Dog();

System.out.println(m instanceof Animal);

System.out.println(d instanceof Mammal);

System.out.println(d instanceof Animal);

}

}
```

```
PS C:\Users\madet\OneDrive\Desktop\javaPrograms> ^C
PS C:\Users\madet\OneDrive\Desktop\javaPrograms>
PS C:\Users\madet\OneDrive\Desktop\javaPrograms>  c:; cd 'c:\Users\madet\OneDrive\Desktop\jav
aPrograms'; & 'C:\Program Files\Java\jre1.8.0_421\bin\java.exe' '-agentlib:jdwp=transport=dt_
socket,server=n,suspend=y,address=localhost:58764' '-cp' 'C:\Users\madet\AppData\Roaming\Code
\User\workspaceStorage\84266f0d6e3a3bcc849fe3e6dff03fb0\redhat.java\jdt_ws\javaPrograms_29296
3c0\bin' 'Dog'
true
true
true
```

POLYMORPHISM

Virtual Methods:

In this section, I will show you how the behaviour of overridden methods in Java allows you to take advantage of polymorphism when designing your classes. We already have discussed method overriding, where a child class can override a method in its parent. An overridden method is essentially hidden in the parent class, and is not invoked unless the child class uses the super keyword within the overriding method.

```java
public class Employee
{
private String name;
private String address;
private int number;
public Employee(String name, String address, int number)
{
System.out.println("Constructing an Employee");
this.name = name;
this.address = address;
this.number = number;
```
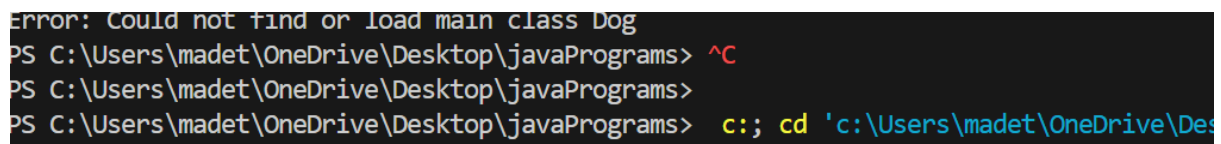
```java
}
public void mailCheck()
{
System.out.println("Mailing a check to " + this.name
+ " " + this.address);
}
public String toString()
{
return name + " " + address + " " + number;
}
public String getName()
{
return name;
}
public String getAddress()
{
return address;
}
public void setAddress(String newAddress)
{
```

```java
address = newAddress;

}

public int getNumber()

{

return number;

}

}
```

Now suppose we extend Employee class as follows:

```java
public class Salary extends Employee

{

private double salary; //Annual salary

public Salary(String name, String address, int number, double

salary)

{

super(name, address, number);

setSalary(salary);

}

public void mailCheck()
```

```java
{
System.out.println("Within mailCheck of Salary class ");
System.out.println("Mailing check to " + getName()
+ " with salary " + salary);
}
public double getSalary()
{
return salary;
}
public void setSalary(double newSalary)
{
if(newSalary >= 0.0)
{
salary = newSalary;
}
}
public double computePay()
{
System.out.println("Computing salary pay for " +
getName());
```

```java
return salary/52;

}

}
```

```
Constructing an Employee
Constructing an Employee
Call mailCheck using Salary reference --
Within mailCheck of Salary class
ailing check to Mohd Mohtashim with salary 3600.0

Call mailCheck using Employee reference--
Within mailCheck of Salary class
ailing check to John Adams with salary 2400.0
```

MULTI THREADING

Java is amulti threaded programming language which means we can develop multi threaded program using Java. A multi threaded program contains two or more parts that can run concurrently and each part can handle different task at the same time making optimal use of the available resources specially when your computer has multiple CPUs.

Above-mentioned stages are explained here:

New:

A new thread begins its life cycle in the new state. It remains in this state until the program starts the thread. It is also referred to as a born thread.

Runnable: After a newly born thread is started, the thread becomes runnable. A thread in this state is considered to be executing its task.

Waiting: Sometimes, a thread transitions to the waiting state while the thread waits for another thread to perform a task.A thread transitions back to the runnable state only when another thread signals the waiting thread to continue executing.

Timed waiting: A runnable thread can enter the timed waiting state for a specified interval of time. A thread in this state transitions back to the runnable state when

that time interval expires or when the event it is waiting for occurs.

Terminated Dead: A runnable thread enters the terminated state when it completes its task or otherwise terminates.

Create Thread by Implementing Runnable Interface:

Step 1:

As a first step you need to implement a run method provided by Runnable interface. This method provides entry point for the thread and you will put you complete business logic inside this method. Following is simple syntax of run method:

```
public void run( )
```

Step 2:

At second step you will instantiate a Thread object using the following constructor:

```
Thread(Runnable threadObj, String threadName);
```

Step 3

Once Thread object is created, you can start it by calling start method, which executes a call to run method. Following is simple syntax of start method:

```
void start( );
```

class RunnableDemo implements Runnable {

    private Thread t;

    private String threadName;

    RunnableDemo(String name){

    threadName = name;

    System.out.println("Creating " + threadName);

    }

    public void run() {

    System.out.println("Running " + threadName);

    try {

    for(int i = 4; i > 0; i--) {

    System.out.println("Thread: " + threadName + ", " + i);

    // Let the thread sleep for a while.

    Thread.sleep(50);

    }

```java
      } catch (InterruptedException e) {

      System.out.println("Thread " + threadName + "
interrupted.");

      }

      System.out.println("Thread " + threadName + "
exiting.");

      }

      public void start ()

      {

      System.out.println("Starting " + threadName );

      if (t == null)

      {

      t = new Thread (this, threadName);

      t.start ();

      }

      }

      }

      public class TestThread {

      public static void main(String args[]) {

      RunnableDemo R1 = new RunnableDemo( "Thread-
1");
```

```
    R1.start();

    RunnableDemo R2 = new RunnableDemo( "Thread-2");

    R2.start();

    }

}
```

```
PS C:\Users\madet\OneDrive\Desktop\javaPrograms>  & 'C:\Program Files\Java\jre1.8.0_421\bin\java.exe
' '-agentlib:jdwp=transport=dt_socket,server=n,suspend=y,address=localhost:59275' '-cp' 'C:\Users\ma
det\AppData\Roaming\Code\User\workspaceStorage\84266f0d6e3a3bcc849fe3e6dff03fb0\redhat.java\jdt_ws\j
avaPrograms_292963c0\bin' 'TestThread'
Creating Thread-1
Starting Thread-1
Creating Thread-2
Starting Thread-2
Running Thread-1
Thread: Thread-1, 4
Running Thread-2
Thread: Thread-2, 4
Thread: Thread-2, 3
Thread: Thread-1, 3
Thread: Thread-2, 2
Thread: Thread-1, 2
Thread: Thread-1, 1
Thread: Thread-2, 1
Thread Thread-1 exiting.
Thread Thread-2 exiting.
PS C:\Users\madet\OneDrive\Desktop\javaPrograms>
```

Create Thread by Extending Thread Class:

The second way to create a thread is to create a new class that extends Thread class using the following two simple steps. This approach provides more flexibility in handling multiple threads created using available methods in Thread class.

Step 1

 You will need to override run method available in Thread class. This method provides entry point for the thread and you will put you complete business logic inside this method. Following is simple syntax of run method:

```
public void run( )
```

Example:

Here is the preceding program rewritten to extend Thread

```
class ThreadDemo extends Thread {

    private Thread t;

    private String threadName;

    ThreadDemo( String name){

    threadName = name;

    System.out.println("Creating " + threadName );

    }

    public void run() {

    System.out.println("Running " + threadName );

    try {

    for(int i = 4; i > 0; i--) {
```

```java
            System.out.println("Thread: " + threadName + ", " +
i);

        // Let the thread sleep for a while.

        Thread.sleep(50);

        }

        } catch (InterruptedException e) {

        System.out.println("Thread " + threadName + "
interrupted.");

        }

        System.out.println("Thread " + threadName + "
exiting.");

    }

    public void start ()

    {

    System.out.println("Starting " + threadName );

    if (t == null)

    {

    t = new Thread (this, threadName);

    t.start ();

    }

    }
```

```java
}
public class TestThread {
public static void main(String args[]) {
ThreadDemo T1 = new ThreadDemo( "Thread-1");
T1.start();
ThreadDemo T2 = new ThreadDemo( "Thread-2");
T2.start();
}
}
```

EXCEPTION

An exception orexceptionalevent is a problem that arises during the execution of a program. When an Exception occurs the normal flow of the program is disrupted and the program/Application terminates abnormally, which is not recommended, therefore these exceptions are to be handled.

An exception can occur for many different reasons, below given are some scenarios where exception occurs.

- 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.

Based on these we have three categories of Exceptions you need to understand them to know how exception handling works in Java

Checked exceptions: A checked exception is an exception that occurs at the compile time, these are

also called as compile time exceptions. These exceptions cannot simply be ignored at the time of compilation, the Programmer should take care of handle these exceptions.

For example, if you use FileReader class in your program to read data from a file, if the file specified in its constructor doesn't exist, then an FileNotFoundException occurs, and compiler prompts the programmer to handle the exception.

CODE:

```java
import java.io.File;
import java.io.FileReader;
public class FilenotFound_Demo {
public static void main(String args[]){
File file=new File("E://file.txt");
FileReader fr = new FileReader(file);
}
}
```

```
PS C:\Users\madet\OneDrive\Desktop\javaPrograms>  & 'C:\Program Files\Java\jre1.8.0_421\bin\j
ava.exe' '-cp' 'C:\Users\madet\AppData\Roaming\Code\User\workspaceStorage\84266f0d6e3a3bcc849
fe3e6dff03fb0\redhat.java\jdt_ws\javaPrograms_292963c0\bin' 'FilenotFound_Demo'
Exception in thread "main" java.lang.Error: Unresolved compilation problem:

        at FilenotFound_Demo.main(Salary.java:4)
PS C:\Users\madet\OneDrive\Desktop\javaPrograms>
```

Exception Hierarchy:

All exception classes are subtypes of the java.lang.Exception class. The exception class is a subclass of the Throwable class. Other than the exception class there is another subclass called Error which is derived from the Throwable class. Errors are not normally trapped form the Java programs. These conditions normally happen in case of severe failures, which are not handled by the java programs. Errors are generated to indicate errors generated by the runtime environment.

Example :

JVM is out of Memory. Normally programs cannot recover from errors. The Exception class has two main subclasses: IOException class and RuntimeException Class.

```
                        ┌─────────────────┐
                        │    java.lang    │
                        └─────────────────┘
                                 │
                                 ▼
                        ┌─────────────────┐
                        │     Object      │
                        └─────────────────┘
                                 │
                                 ▼
                        ┌─────────────────┐
                        │    Throwable    │
                        └─────────────────┘
                    ┌────────────┴──────────────────────┐
                    ▼                                    ▼
           ┌─────────────────┐              ┌─────────────────┐
           │     Errors      │              │   Exceptions    │
           └─────────────────┘              └─────────────────┘
                                      ┌────────────┴────────────┐
                                      ▼                         ▼
                          ┌────────────────────┐   ┌────────────────────┐
                          │ Runtime Exceptions │   │  Other Exceptions  │
                          └────────────────────┘   └────────────────────┘
```
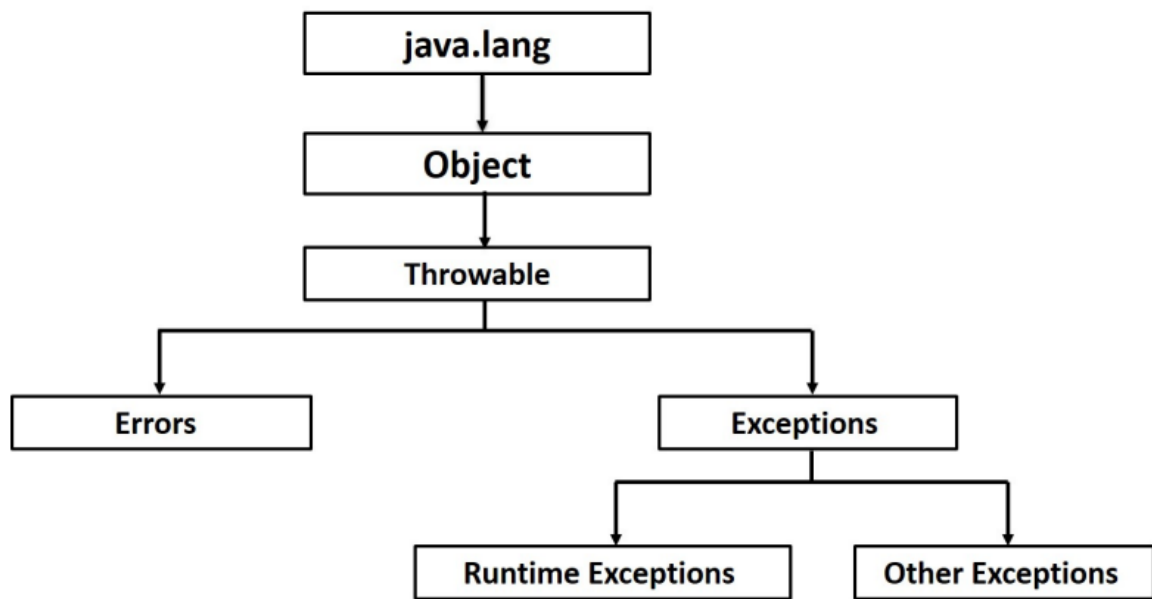
Catching Exceptions:

 A method catches an exception using a combination of the try and catch keywords. A try/catch block is placed around the code that might generate an exception. Code within a try/catch block is referred to as protected code, and the syntax for using try/catch looks like the following:

CODE:

```java
import java.io.*;
public class CheckingAccount
{
private double balance;
private int number;
public CheckingAccount(int number)
```

```java
{
this.number = number;
}
public void deposit(double amount)
{
balance += amount;
}
public void withdraw(double amount) throws InsufficientFundsException
{
if(amount <= balance)
{
balance -= amount;
}
else
{
double needs = amount - balance;
throw new InsufficientFundsException(needs);
}
}
```

```java
public double getBalance()

{

return balance;

}

public int getNumber()

{

return number;

}

}
```

```
Depositing $500...

Withdrawing $100...

Withdrawing $600...
Sorry, but you are short $200.0
InsufficientFundsException
        at CheckingAccount.withdraw(CheckingAccount.java:25)
        at BankDemo.main(BankDemo.java:13)
```

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.

A "Hello, World" Applet:

EVENT HANDLING

What is an Event?

Change in the state of an object is known as event i.e. event describes the change in state of source. Events are generated as result of user interaction with the graphical user interface components. For example, clicking on a button, moving the mouse, entering a character through keyboard,selecting an item from list, scrolling the page are the activities that causes an event to happen.

Types of Event The events can be broadly classified into two categories:

Foreground Events –

Those events which require the direct interaction of user.They are generated as consequences of a person interacting with the graphical components in Graphical User Interface. For example, clicking on a button, moving the mouse, entering a character through keyboard,selecting an item from list, scrolling the page etc.

Background Events –

Those events that require the interaction of end user are known as background events. Operating system

interrupts, hardware or software failure, timer expires, an operation completion are the example of background events.

What is Event Handling?

Event Handling is the mechanism that controls the event and decides what should happen if an event occurs. This mechanism have the code which is known as event handler that is executed when an event occurs. Java Uses the Delegation Event Model to handle the events.

Steps involved in event handling

- The User clicks the button and the event is generated.
- Now the object of concerned event class is created automatically and information about the source and the event get populated with in same object.
- Event object is forwarded to the method of registered listener class.
- the method is now get executed and returns.

Points to remember about listener

- In order to design a listener class we have to develop some listener interfaces.These Listener interfaces forecast some public

abstract callback methods which must be implemented by the listener class.

- If you do not implement the any if the predefined interfaces then your class can not act as a listener class for a source object.

CODE:

```java
package javaprogram;

import java.awt.*;

import java.awt.event.*;

import javax.swing.*;

public class EventHandling {
  private JFrame mainFrame;

  private JLabel headerLabel;

  private JLabel statusLabel;

  private JPanel controlPanel;

  public EventHandling(){

    prepareGUI();

  }

  public static void main(String[] args){

    EventHandling swingControlDemo = new EventHandling();
```
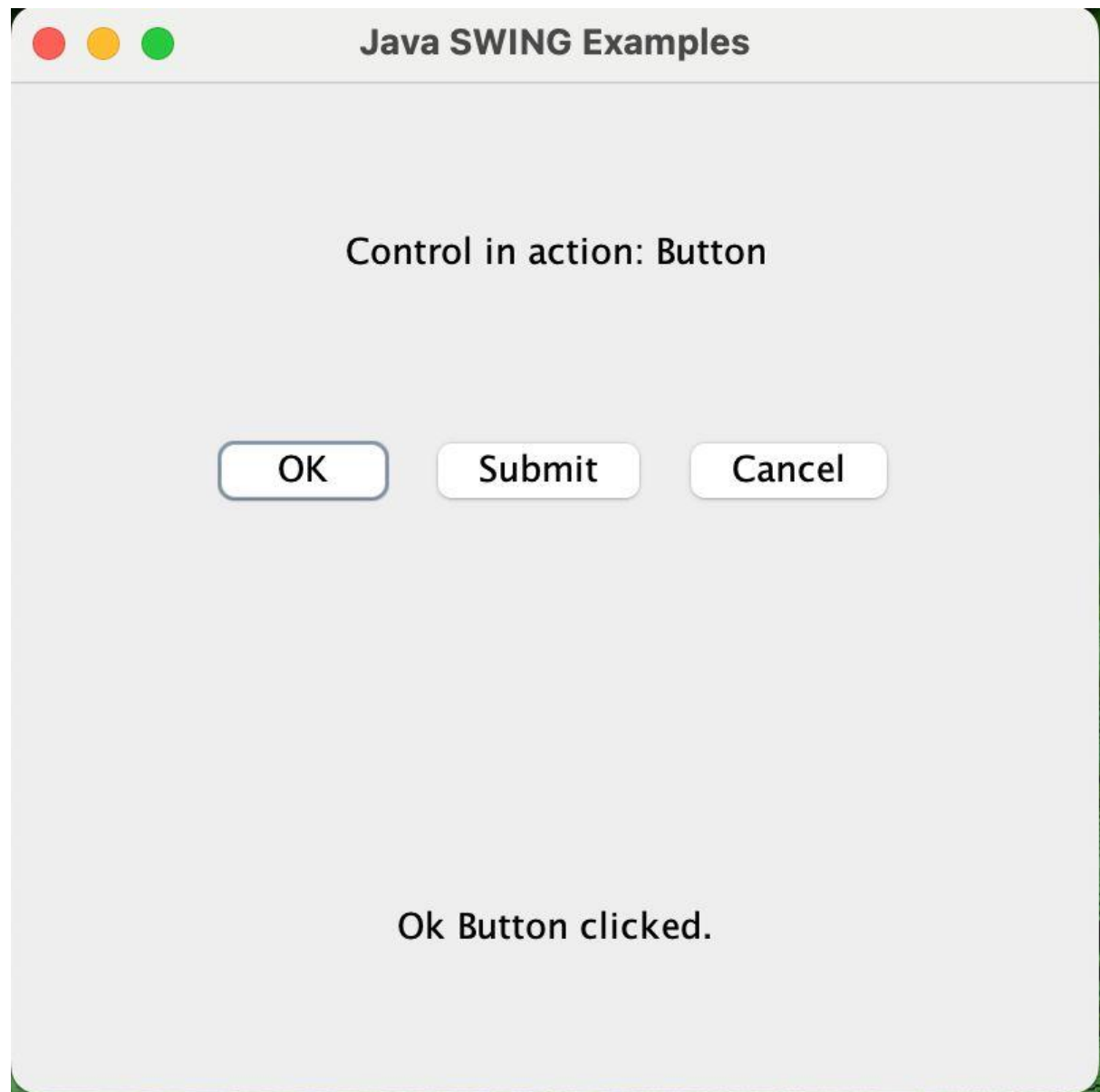
```java
      swingControlDemo.showEventDemo();
   }

   private void prepareGUI(){
      mainFrame = new JFrame("Java SWING Examples");
      mainFrame.setSize(400,400);
      mainFrame.setLayout(new GridLayout(3, 1));
      headerLabel = new JLabel("",JLabel.CENTER );
      statusLabel = new JLabel("",JLabel.CENTER);
      statusLabel.setSize(350,100);
      mainFrame.addWindowListener(new
WindowAdapter() {
         public void windowClosing(WindowEvent
windowEvent){
            System.exit(0);
         }
      });
      controlPanel = new JPanel();
      controlPanel.setLayout(new FlowLayout());
      mainFrame.add(headerLabel);
      mainFrame.add(controlPanel);
```

```java
        mainFrame.add(statusLabel);

        mainFrame.setVisible(true);

    }

    private void showEventDemo(){

        headerLabel.setText("Control in action: Button");

        JButton okButton = new JButton("OK");

        JButton submitButton = new JButton("Submit");

        JButton cancelButton = new JButton("Cancel");

        okButton.setActionCommand("OK");

        submitButton.setActionCommand("Submit");

        cancelButton.setActionCommand("Cancel");

        okButton.addActionListener(new
ButtonClickListener());

        submitButton.addActionListener(new
ButtonClickListener());

        cancelButton.addActionListener(new
ButtonClickListener());

        controlPanel.add(okButton);

        controlPanel.add(submitButton);

        controlPanel.add(cancelButton);

        mainFrame.setVisible(true);
```

```java
    }
    private class ButtonClickListener implements
ActionListener{
        public void actionPerformed(ActionEvent e) {
            String command = e.getActionCommand();
            if( command.equals( "OK" )) {
                statusLabel.setText("Ok Button clicked.");
            }
            else if( command.equals( "Submit" ) ) {
                statusLabel.setText("Submit Button clicked.");
            }
            else {
                statusLabel.setText("Cancel Button clicked.");
            }
        }
    }
}
```

Java SWING Examples

Control in action: Button

OK	Submit	Cancel

Ok Button clicked.

FILES AND I/O

Stream A stream can be defined as a sequence of data. there are two kinds of Streams

InPutStream: The InputStream is used to read data from a source.

OutPutStream: the OutputStream is used for writing data to a destination.

Byte streams:

```
import java.io.*;

public class CopyFile {
    public static void main(String[] args) throws IOException {
        FileInputStream in = null;
        FileOutputStream out = null;
        try {
```

```java
        in = new FileInputStream("input.txt");

        out = new FileOutputStream("output.txt");

        int c;

        while ((c = in.read()) != -1) {

            out.write(c);

        }

    } finally {

        if (in != null) {

            in.close();

        }

        if (out != null) {

            out.close();

        }

    }

  }
}
```

Output: