

# LAB MANUAL



## CSC 103L OBJECT ORIENTED PROGRAMMING (v. 1.1)

|                           |  |
|---------------------------|--|
| <b>Student Name:</b>      |  |
| <b>Class and Section:</b> |  |
| <b>Roll Number:</b>       |  |
| <b>CGPA:</b>              |  |
| <b>Email ID:</b>          |  |

**DEPARTMENT OF SOFTWARE ENGINEERING**  
**Foundation University Rawalpindi Campus (FURC)**  
**New Lalazar, Rawalpindi, Pakistan**  
[www.fui.edu.pk](http://www.fui.edu.pk)

## **PREFACE**

This lab manual has been prepared to facilitate the students of software engineering/computer science in studying and analysing object orientation. Object Orientation is a technique used extensively in software engineering to build software products as a bottom-up method. The object orientation allows us to view the world as objects having attributes and methods. To perform a task objects, communicate with other objects by invoking methods.

## **PREPARED BY**

Lab manual is prepared by Engr. Umar Mahmud under the supervision of Head of Department Dr. M. Shaheen Khan in the year 2016. The manual was further revised by Dr. Shariq, Mr. M. Ishtiaq and Mr. M. Fahad

## **GENERAL INSTRUCTIONS**

- a. Students are required to maintain the lab manual with them till the end of the semester.
- b. All readings, answers to questions and illustrations must be solved on the place provided. If more space is required then additional sheets may be attached.
- c. It is the responsibility of the student to have the manual graded before deadlines as given by the instructor
- d. Loss of manual will result in re-submission of the complete manual.
- e. Students are required to go through the experiment before coming to the lab session. Lab session details will be given in training schedule.
- f. Students must bring the manual in each lab.
- g. Keep the manual neat clean and presentable.
- h. Plagiarism is strictly forbidden. No credit will be given if a lab session is plagiarised and no re submission will be entertained.
- i. Marks will be deducted for late submission.
- j. Error handling in a program is the responsibility of the Student.

## **VERSION HISTORY**

| <b>Date</b>  | <b>Update By</b>                         | <b>Details</b>   |
|--------------|--|--|
| January 2016 | Engr. Umar Mahmud                        | First Version Created  |
| January 2017 | Dr. Shariq, Mr. M. Ishtiaq, Mr. M. Fahad | Version 1.1. Project outline is included.  |
| August 2019  | Dr. Muhammad Habib, Ms. Fauzia Khan      | Version 2.0. Method overloading, Term project, Suggested projects list and a sample project is included. |
|              |  |  |
|              |  |  |

## MARKS

| Exp #             | Date Conducted | Experiment Title              | Max. Marks | Marks Obtained | Instructor Sign |
|-------------------|----------------|-------------------------------|------------|----------------|-----------------|
| 1                 |                | INTRODUCTION TO JAVA          | 10         |                |                 |
| 2                 |                | CONTROL STRUCTURES AND LOOPS  | 10         |                |                 |
| 3                 |                | METHODS                       | 10         |                |                 |
| 4                 |                | CLASSES AND OBJECTS           | 10         |                |                 |
| 5                 |                | INHERITANCE                   | 10         |                |                 |
| 6                 |                | OVERLOADING AND OVERRIDING    | 10         |                |                 |
| 7                 |                | ABSTRACTION                   | 10         |                |                 |
| 8                 |                | ENCAPSULATION                 | 10         |                |                 |
| 9                 |                | INTERFACES                    | 10         |                |                 |
| 10                |                | JAVA PACKAGES                 | 10         |                |                 |
| 11                |                | POLYMORPHISM                  | 10         |                |                 |
| 12                |                | GUI USING SWING IN JAVA       | 10         |                |                 |
| 13                |                | EXCEPTION HANDLING            | 10         |                |                 |
| 14                |                | PAINT APPLICATION IN NETBEANS | 10         |                |                 |
| SESSIONAL TOTAL   |                |                               |            |                |                 |
|                   |                | PROJECT PHASE - I             | 10         |                |                 |
|                   |                | PROJECT PHASE - II            | 20         |                |                 |
| PROJECT TOTAL     |                |                               | 30         |                |                 |
| GRAND TOTAL (100) |                |                               | 100        |                |                 |

## **LIST OF EXPERIMENTS**

|  |    |
|--|----|
| EXPERIMENT 1 – INTRODUCTION TO JAVA                      | 5  |
| EXPERIMENT 2 – CONTROL STRUCTURES AND LOOPS              | 11 |
| EXPERIMENT 3 – METHODS                                   | 17 |
| EXPERIMENT 4 – CLASSES AND OBJECTS                       | 21 |
| EXPERIMENT 5 – INHERITANCE                               | 25 |
| EXPERIMENT 6 – OVERRIDING                                | 29 |
| EXPERIMENT 7 – ABSTRACTION                               | 36 |
| EXPERIMENT 8 – ENCAPSULATION                             | 41 |
| EXPERIMENT 9 – INTERFACES                                | 44 |
| EXPERIMENT 10 – JAVA PACKAGES                            | 48 |
| EXPERIMENT 11 – POLYMORPHISM                             | 52 |
| EXPERIMENT 12 – GUI USING SWING IN JAVA                  | 56 |
| EXPERIMENT 13 – EXCEPTION HANDLING                       | 75 |
| EXPERIMENT 14 – PAINT APPLICATION IN JAVA USING NETBEANS | 83 |
| PROJECT –OBJECT ORIENTED PROGRAMMING - PHASE - I         | 98 |
| PROJECT –OBJECT ORIENTED PROGRAMMING - PHASE - II        | 99 |

## EXPERIMENT 1 – INTRODUCTION TO JAVA

1. **Objectives:**

- (a). Learning Java Language
- (b). Learning to program in Java
- (c). Executing Java programs
- (d). Java variables
- (e). Arithmetic Expressions

2. **Time Required:** 3 hrs

3. **Programming Language:** Java

4. **Software Required:**

- (a). Windows OS
- (b). NetBeans / Eclipse Kepler

5. **Java** is a high-level programming language originally developed by Sun Microsystems and released in 1995. Java runs on a variety of platforms, such as Windows, Mac OS, and the various versions of UNIX.

Java is a general-purpose, concurrent, class-based, object-oriented computer programming language that is specifically designed to have as few implementation dependencies as possible. It is intended to let application developers "write once, run anywhere" (WORA), meaning that code that runs on one platform does not need to be recompiled to run on another.

Java applications are typically compiled to bytecode (class file) that can run on any Java virtual machine (JVM) regardless of computer architecture.

Java is, as of 2017, one of the most popular programming languages in use, particularly for client-server web applications. The language derives much of its syntax from C and C++, but it has fewer low-level facilities than either of them.

6. **Java Compiler** is software that compiles the Java code into Bytecode. Bytecode is a set of commands that the java virtual machine understands. The Java Compiler is open source and is called as the Java Standard Development Kit (SDK). Currently version 8 is available online. The SDK must be available before editing or creating a Java program.

5. **Java Runtime Environment (JRE)** is the virtual machine over which the java code is executed. This virtual machine executes the java code. The JRE must be provided before executing a Java program.

6. **Java Integrated Development Environment (IDE)** is an editor tool complete with Java SDK and JRE and an editing tool that allows a developer to create edit and execute Java programs. Different IDEs are available including NetBeans, Eclipse, Kawa, etc.

7. To develop Java programs, you will need to download the IDE. This lab follows Netbeans which can be acquired from <https://netbeans.org/downloads/>. Download the Netbeans with JDK for Windows (64 bit) and install the software. <https://netbeans.org/downloads/>

8. After downloading execute the software and be acquainted with its environment (class activity)
9. A Java program is created as a Java Class. A Class is an abstract data type that has attributes that describe the class as well as functions that allow the access and manipulation of these attributes.
10. What is the difference between a Class and a Structure in C/C++? (1)
11. To create a Java class, remember that the file name should be same as the class name. Also only one class can be created in a Java file.
12. Create a code in C++ that displays 'Hello World' (without quotes). Execute the code and show output here. (5)

13. The following creates a Java class 'MyFirstJavaProgram' that has a function main. The main function calls a utility in System class that displays the string 'Hello World'

```
public class MyFirstJavaProgram {  
  
    public static void main(String []args) {  
        System.out.println("Hello World");  
    }  
}
```

The Java naming convention requires that the Class names should be without spaces and the first letter of each word should be capitalized.

14. In Netbeans create a new file with name MyFirstJavaProgram. Type the code here (5) and execute it. Display the output here.

15. What is the difference in the code written by you in Point 12 and the code (1)  
provided in point 14?

16. **Variables** are constructs that allocate a memory location in RAM and place values in it. The capability of a variable is the ability to vary. Variables are created as: -

```
int x = 0;  
char c = 'c';  
float myFloat = 3.14;  
General format is:
```

```
type identifier [ = value][, identifier [= value] ...] ;
```

Data types in java are given here  
[http://www.tutorialspoint.com/java/java\\_basic\\_datatypes.htm](http://www.tutorialspoint.com/java/java_basic_datatypes.htm)

The naming convention is the same as that of C++.

```
int a, b, c;           // declares three ints, a, b, and c.  
int d = 3, e, f = 5;  // declares three more ints, initializing  
                      // d and f.  
byte z = 22;          // initializes z.  
double pi = 3.14159;  // declares an approximation of pi.  
char x = 'x';         // the variable x has the value 'x'.
```

17. **String** is a special construct that is an array of type char. A string is created in Java as

**String s = "OOP";**

18. Update the program in Point 13 and create a string variable of your name. Edit the (1)  
program so that your name is printed. Show output and code here.

19. **Arithmetic Expressions** are a construct made up of variables and operators which are constructed according to the syntax of the language that evaluates to a single value.

**int result = 1 + 2; // result is now 3**

The expressions are same as in C/C++..Java arithmetic operators are:

| Operator | Description   | Example             |
|----------|---|---------------------|
| +        | Addition - Adds values on either side of the operator                           | A + B will give 30  |
| -        | Subtraction - Subtracts right hand operand from left hand operand               | A - B will give -10 |
| *        | Multiplication - Multiplies values on either side of the operator               | A * B will give 200 |
| /        | Division - Divides left hand operand by right hand operand                      | B / A will give 2   |
| %        | Modulus - Divides left hand operand by right hand operand and returns remainder | B % A will give 0   |
| ++       | Increment - Increase the value of operand by 1                                  | B++ gives 21        |
| --       | Decrement - Decrease the value of operand by 1                                  | B-- gives 19        |

The Relational operators are

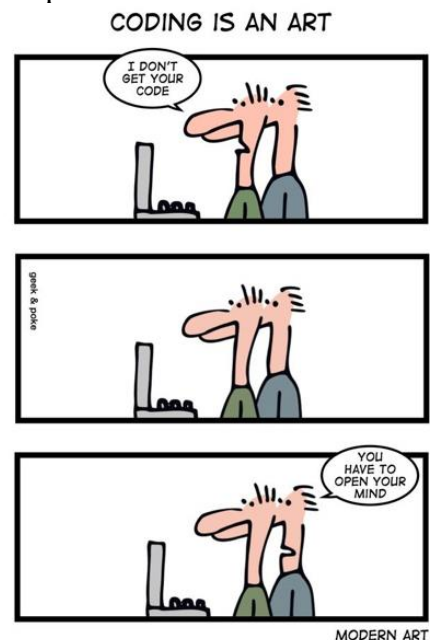
| Operator | Description   | Example               |
|----------|---|-----------------------|
| ==       | Checks if the value of two operands are equal or not, if yes then condition becomes true.                                       | (A == B) is not true. |
| !=       | Checks if the value of two operands are equal or not, if values are not equal then condition becomes true.                      | (A != B) is true.     |
| >        | Checks if the value of left operand is greater than the value of right operand, if yes then condition becomes true.             | (A > B) is not true.  |
| <        | Checks if the value of left operand is less than the value of right operand, if yes then condition becomes true.                | (A < B) is true.      |
| >=       | Checks if the value of left operand is greater than or equal to the value of right operand, if yes then condition becomes true. | (A >= B) is not true. |
| <=       | Checks if the value of left operand is less than or equal to the value of right operand, if yes then condition becomes true.    | (A <= B) is true.     |

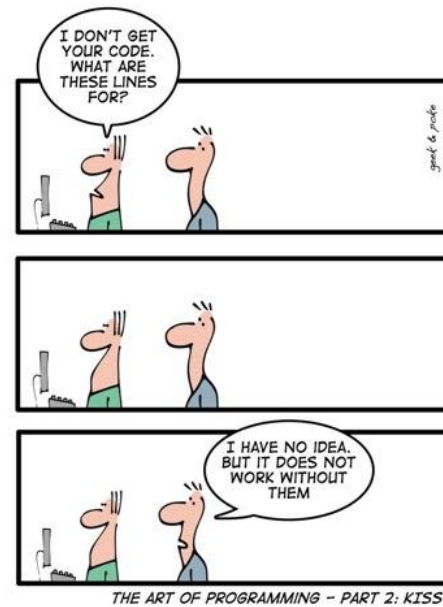
The logical operators are: -

| Operator | Description  | Example            |
|----------|--|--------------------|
| &&       | Called Logical AND operator. If both the operands are non zero then then condition becomes true.   | (A && B) is false. |
|          | Called Logical OR Operator. If any of the two operands are non zero then then condition becomes true.  | (A    B) is true.  |
| !        | Called Logical NOT Operator. Use to reverses the logical state of its operand. If a condition is true then Logical NOT operator will make false. | !(A && B) is true. |



20. Update the code in Point 19 that displays the result of the expression in Point 20. (1)  
Show code and output here
21. Create a Java code that declares two variables one of which is PI. Keep the value of PI at 3.14. Calculate the area of a circle given the radius and display it. Write code and output for radii 2, 4 and 5 (2)
22. Create a Java code that for numbers 2, 3 and 4 calculates the half of these numbers. Display the halves and write code and output here. (2)





### **Web Resources**

<https://netbeans.org/downloads/>

<http://www.oracle.com/technetwork/java/javase/downloads/jdk-7-netbeans-download-432126.html>

<http://www.learnjavaonline.org/>

<http://docs.oracle.com/javase/tutorial/>

### **Video Resource**

<http://www.youtube.com/watch?v=KheL6umdW-s>

<http://www.youtube.com/watch?v=3MZIkY55fS0>

<http://www.learnerstv.com/Free-Computers-Video-lectures-ltv006-Page1.htm>

<http://www.youtube.com/watch?v=Hv2yvXTVTVo>

**Summary:** This is the basic tutorial of Java that allows the student to create basic Java programs similar to C++/C. The programs include variables, arithmetic statements and display functions.

## EXPERIMENT 2 – CONTROL STRUCTURES AND LOOPS

1. **Objectives:**
  - (a). Conditionals in Java
  - (b). Arrays and Strings
  - (c). Loops in Java
2. **Time Required:** 3 hrs
3. **Programming Language:** Java
4. **Software Required:**
  - (a). Windows OS
  - (b). NetBeans / Eclipse Kepler
5. **Decision Making:** There are two types of decision making statements in Java. They are: if statements and switch statements.
6. **IF Statements:** An **if** statement consists of a Boolean expression followed by (.5) one or more statements.

```
public class Test {  
  
    public static void main(String args[]){  
        int x = 10;  
  
        if( x < 20 ){  
            System.out.print("This is if statement");  
        }  
    }  
}
```

Execute the code and print the output here:

7. **IF ELSE Statements:** (1)

```
public class Test {  
  
    public static void main(String args[]){  
        int x = 30;  
  
        if( x < 20 ){  
            System.out.print("This is if statement");  
        }else{  
            System.out.print("This is else statement");  
        }  
    }  
}
```

What will be the output for the following cases (show output): -

(a).  $x = 20$

(b).  $x = 19$

8. **Strings:** Strings, which are widely used in Java programming, are a sequence of characters. In the Java programming language, strings are objects. The Java platform provides the String class to create and manipulate strings. The most direct way to create a string is to write:

```
String greeting = "Hello world!";
```

Methods used to obtain information about an object are known as accessor methods. One accessor method that you can use with strings is the length() method, which returns the number of characters contained in the string object.

```
public class StringDemo {  
  
    public static void main(String args[]) {  
        String palindrome = "Dot saw I was Tod";  
        int len = palindrome.length();  
        System.out.println( "String Length is : " + len );  
    }  
}
```

The String class includes a method for concatenating two strings:

```
string1.concat(string2);
```

9. **Arrays:** Java provides a data structure, the **array**, which stores a fixed-size sequential collection of elements of the same type. An array is used to store a collection of data, but it is often more useful to think of an array as a collection of variables of the same type. Instead of declaring individual variables, such as number0, number1, ..., and number99, you declare one array variable such as numbers and use numbers[0], numbers[1], and ..., numbers[99] to represent individual variables. To use an array in a program, you must declare a variable to reference the array, and you must specify the type of array the variable can reference. Here is the syntax for declaring an array variable:

```
double[] myList;           // preferred way.

or

double myList[];          // works but not preferred way.
```

You can create an array by using the new operator with the following syntax:

```
arrayRefVar = new dataType[arraySize];
```

The above statement does two things:

- (a). It creates an array using new dataType[arraySize];
- (b). It assigns the reference of the newly created array to the variable arrayRefVar.

10. **Processing Arrays:** Execute this program and show the output here (5)

```
public class TestArray {

    public static void main(String[] args) {
        double[] myList = {1.9, 2.9, 3.4, 3.5};

        // Print all the array elements
        for (int i = 0; i < myList.length; i++) {
            System.out.println(myList[i] + " ");
        }
        // Summing all elements
        double total = 0;
        for (int i = 0; i < myList.length; i++) {
            total += myList[i];
        }
        System.out.println("Total is " + total);
        // Finding the largest element
        double max = myList[0];
        for (int i = 1; i < myList.length; i++) {
            if (myList[i] > max) max = myList[i];
        }
        System.out.println("Max is " + max);
    }
}
```

11. **Loops:** There may be a situation when we need to execute a block of code several number of times, and is often referred to as a loop. Java has very flexible three looping mechanisms. You can use one of the following three

loops:

- (a). while Loop
- (b). do...while Loop
- (c). for Loop

12. **While Loop:** Execute the following code and write the output. (.5)

```
public class Test {  
  
    public static void main(String args[]) {  
        int x = 10;  
  
        while( x < 20 ) {  
            System.out.print("value of x : " + x );  
            x++;  
            System.out.print("\n");  
        }  
    }  
}
```

13. **Do While Loop:** Execute the following code and write the output (.5)

```
public class Test {  
  
    public static void main(String args[]){  
        int x = 10;  
  
        do{  
            System.out.print("value of x : " + x );  
            x++;  
            System.out.print("\n");  
        }while( x < 20 );  
    }  
}
```

14. **For Loop:** Execute the following code and write the output (5)

```
public class Test {  
  
    public static void main(String args[]) {  
  
        for(int x = 10; x < 20; x = x+1) {  
            System.out.print("value of x : " + x );  
            System.out.print("\n");  
        }  
    }  
}
```

15. Is the syntax of loops in Java same as that in C/C++? Yes/No (5)
16. What is the difference between While loop and Do While loop? (1)
17. Create a code that stores the first 10 prime numbers in an array. Then using arithmetic, in a loop of your choice, calculate the square and the cube of each prime number and display appropriately. Show code and output here. (5)

**Web Resources:**

[http://www.tutorialspoint.com/java/java\\_loop\\_control.htm](http://www.tutorialspoint.com/java/java_loop_control.htm)

[http://www.tutorialspoint.com/java/java\\_arrays.htm](http://www.tutorialspoint.com/java/java_arrays.htm)

[http://www.tutorialspoint.com/java/java\\_strings.htm](http://www.tutorialspoint.com/java/java_strings.htm)

**Video Resource:**

<http://www.learnerstv.com/Free-Computers-Video-lectures-ltv006-Page1.htm>

**Summary:** This lab session introduces arrays strings, conditionals and loops in Java. This lab is designed to give an experience to students to create and write basic java codes.



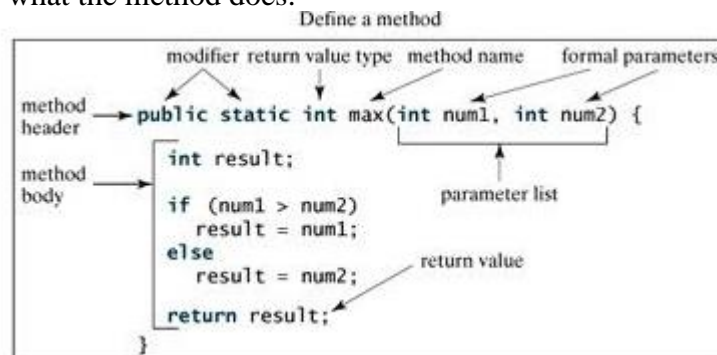
## EXPERIMENT 3 – METHODS

1. **Objectives:**
  - (a). Methods in Java
  - (b). Parameter passing and return statements
  - (c). Method invocations
2. **Time Required:** 3 hrs
3. **Programming Language:** Java
4. **Software Required:**
  - (a). Windows OS
  - (b). NetBeans / Eclipse Kepler
5. **Method:** A Java method is a collection of statements that are grouped together to perform an operation. When you call the System.out.println method, for example, the system actually executes several statements in order to display a message on the console. In general, a method has the following syntax:

```
modifier returnType methodName(list of parameters) {  
    // Method body;  
}
```

A method definition consists of a method header and a method body. Here are all the parts of a method:

- (a). **Modifiers:** The modifier, which is optional, tells the compiler how to call the method. This defines the access type of the method.
- (b). **Return Type:** A method may return a value. The returnType is the data type of the value the method returns. Some methods perform the desired operations without returning a value. In this case, the returnType is the keyword **void**.
- (c). **Method Name:** This is the actual name of the method. The method name and the parameter list together constitute the method signature.
- (d). **Parameters:** A parameter is like a placeholder. When a method is invoked, you pass a value to the parameter. This value is referred to as actual parameter or argument. The parameter list refers to the type, order, and number of the parameters of a method. Parameters are optional; that is, a method may contain no parameters.
- (e). **Method Body:** The method body contains a collection of statements that define what the method does.



6. In creating a method, you give a definition of what the method is to do. To use a

method, you have to call or invoke it. There are two ways to call a method; the choice is based on whether the method returns a value or not. When a program calls a method, program control is transferred to the called method. A called method returns control to the caller when its return statement is executed or when its method-ending closing brace is reached. If the method returns a value, a call to the method is usually treated as a value. Execute the given code and write output here. Also trace the control flow on the code using a pencil (1)

```
public class TestMax {  
    /** Main method */  
    public static void main(String[] args) {  
        int i = 5;  
        int j = 2;  
        int k = max(i, j);  
        System.out.println("The maximum between " + i +  
                           " and " + j + " is " + k);  
    }  
  
    /** Return the max between two numbers */  
    public static int max(int num1, int num2) {  
        int result;  
        if (num1 > num2)  
            result = num1;  
        else  
            result = num2;  
  
        return result;  
    }  
}
```

7. Following is a program that demonstrates the effect of passing by value. The program creates a method for swapping two variables. The swap method is invoked by passing two arguments. Interestingly, the values of the arguments are not changed after the method is invoked. Execute the code and draw trace call (1)

```

public class TestPassByValue {

    public static void main(String[] args) {
        int num1 = 1;
        int num2 = 2;

        System.out.println("Before swap method, num1 is " +
            num1 + " and num2 is " + num2);

        // Invoke the swap method
        swap(num1, num2);
        System.out.println("After swap method, num1 is " +
            num1 + " and num2 is " + num2);
    }
    /** Method to swap two variables */
    public static void swap(int n1, int n2) {
        System.out.println("\tInside the swap method");
        System.out.println("\t\tBefore swapping n1 is " + n1
            + " n2 is " + n2);

        // Swap n1 with n2
        int temp = n1;
        n1 = n2;
        n2 = temp;

        System.out.println("\t\tAfter swapping n1 is " + n1
            + " n2 is " + n2);
    }
}

```

8. Create a function that when given an integer value returns the square of the function. Write the code and show the output here. Show the trace call on the code using a pencil. (1)

9. Write a function that accepts two strings and returns the longer of both. Display the longer string. Write the code and show the output here. Show the trace call on the code using a pencil. (3)
10. Write a function that receives an array and displays the contents of the array. Write the code and show the output here. Show the trace call on the code using a pencil.(2)

**Web Resources:**

[http://www.tutorialspoint.com/java/java\\_methods.htm](http://www.tutorialspoint.com/java/java_methods.htm)

**Video Resources:**

<http://www.learnerstv.com/Free-Computers-Video-lectures-ltv006-Page1.htm>

**Summary:** This lab is designed to create methods in Java. Students are given exercise questions to practice method invocations and parameter passing for different problems.

## EXPERIMENT 4 – CLASSES AND OBJECTS

1. **Objectives:**
  - (a). Creating classes and Objects
  - (b). Attributes and members of a class
2. **Time Required:** 3 hrs
3. **Programming Language:** Java
4. **Software Required:**
  - (a). Windows OS
  - (b). NetBeans / Eclipse Kepler
5. **Object:** Objects have states and behaviors. Example: A dog has states-color, name, breed as well as behaviors -wagging, barking and eating. An object is an instance of a class. If we consider the real-world we can find many objects around us, Cars, Dogs, Humans etc. All these objects have a state and behavior. If we consider a dog then its state is - name, breed, color, and the behavior is - barking, wagging, running.  
If you compare the software object with a real world object, they have very similar characteristics. Software objects also have a state and behavior. A software object's state is stored in attributes and behavior is shown via methods. So in software development methods operate on the internal state of an object and the object-to-object communication is done via methods.
6. **Class:** A class can be defined as a template/ design that describe the behaviors/states that object of its type support.

```
public class Dog{  
    String breed;  
    int age;  
    String color;  
  
    void barking(){  
    }  
  
    void hungry(){  
    }  
  
    void sleeping(){  
    }  
}
```

A class can contain any of the following variable types.

- (a). **Local variables:** variables defined inside methods, constructors or blocks are called local variables. The variable will be declared and initialized within the method and the variable will be destroyed when the method has completed.
- (b). **Instance variables:** Instance variables are variables within a class but outside any method. These variables are instantiated when the class is loaded. Instance variables can be accessed from inside any method, constructor or blocks of that particular class.
- (c). **Class variables:** Class variables are variables declared with in a class, outside any method, with the static keyword.

A class can have any number of methods to access the value of various kind of

methods. In the above example, barking(), hungry() and sleeping() are methods.

7. **Constructors:** When discussing about classes one of the most important sub topic would be constructors. Every class has a constructor. If we do not explicitly write a constructor for a class the java compiler builds a default constructor for that class. Each time a new object is created at least one constructor will be invoked. The main rule of constructors is that they should have the same name as the class. A class can have more than one constructor.

```
public class Puppy{
    public puppy(){
    }

    public puppy(String name){
        // This constructor has one parameter, name.
    }
}
```

8. **Creating an Object:** As mentioned previously a class provides the design for objects. So basically an object is created from a class. In java the new key word is used to create new objects.

There are three steps when creating an object from a class:

- (a). **Declaration** - A variable declaration with a variable name with an object type.
- (b). **Instantiation** - The 'new' key word is used to create the object.
- (c). **Initialization** - The 'new' keyword is followed by a call to a constructor. This call initializes the new object.

```
public class Puppy{

    public Puppy(String name){
        // This constructor has one parameter, name.
        System.out.println("Passed Name is :" + name );
    }

    public static void main(String []args){
        // Following statement would create an object myPuppy
        Puppy myPuppy = new Puppy( "tommy" );
    }
}
```

9. Compile and execute the following code. Show the output here (1)

```

public class Puppy{

    int puppyAge;

    public Puppy(String name){
        // This constructor has one parameter, name.
        System.out.println("Passed Name is :" + name );
    }

    public void setAge( int age ){
        puppyAge = age;
    }

    public int getAge( ){
        System.out.println("Puppy's age is :" + puppyAge );
        return puppyAge;
    }

    public static void main(String []args){
        /* Object creation */
        Puppy myPuppy = new Puppy( "tommy" );

        /* Call class method to set puppy's age */
        myPuppy.setAge( 2 );

        /* Call another class method to get puppy's age */
        myPuppy.getAge( );

        /* You can access instance variable as follows as well */
        System.out.println("Variable Value :" + myPuppy.puppyAge );
    }
}

```

10. What happens in the code in Point 9? Explain in detail. Also identify how is (9) OOP different from procedural programming?

**Web Resources:**

[http://www.tutorialspoint.com/java/java\\_object\\_classes.htm](http://www.tutorialspoint.com/java/java_object_classes.htm)

<http://docs.oracle.com/javase/tutorial/java/javaOO/classes.html>

**Video Resources:**

<http://www.learnerstv.com/Free-Computers-Video-lectures-ltv006-Page1.htm>

**Summary:** This lab introduces the concept of classes and objects and provides practice in creating objects as well as calling member functions.



## EXPERIMENT 5 – INHERITANCE

1. **Objectives:**
  - (a). Inheritance
  - (b). Implementing inheritance in Java
2. **Time Required:** 3 hrs
3. **Programming Language:** Java
4. **Software Required:**
  - (a). Windows OS
  - (b). NetBeans / Eclipse Kepler
5. **Inheritance** can be defined as the process where one object acquires the properties of another. With the use of inheritance the information is made manageable in a hierarchical order. When we talk about inheritance the most commonly used keyword would be **extends** and **implements**. These words would determine whether one object IS-A type of another. By using these keywords we can make one object acquire the properties of another object.
6. **IS-A Relation:** IS-A is a way of saying: This object is a type of that object. Let us see how the **extends** keyword is used to achieve inheritance.

```
public class Animal{  
}  
  
public class Mammal extends Animal{  
}  
  
public class Reptile extends Animal{  
}  
  
public class Dog extends Mammal{  
}
```

Now based on the above example, In Object Oriented terms following are true:

- (a). Animal is the superclass of Mammal class.
- (b). Animal is the superclass of Reptile class.
- (c). Mammal and Reptile are sub classes of Animal class.
- (d). Dog is the subclass of both Mammal and Animal classes.

Now if we consider the IS-A relationship we can say:

- (a). Mammal IS-A Animal
- (b). Reptile IS-A Animal
- (c). Dog IS-A Mammal
- (d). Hence : Dog IS-A Animal as well

With use of the **extends** keyword the subclasses will be able to inherit all the properties of the superclass except for the private properties of the superclass. We can assure that Mammal is actually an Animal with the use of the instance operator.

7. Execute the following code and show output here. What happened in this code? (3)

```

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);

    }

}

```

8. **HAS-A Relationship**: These relationships are mainly based on the usage. This determines whether a certain class HAS-A certain thing. This relationship helps to reduce duplication of code as well as bugs.

```

public class Vehicle{}
public class Speed{}
public class Van extends Vehicle{
    private Speed sp;
}

```

This shows that class Van HAS-A Speed. By having a separate class for Speed we do not have to put the entire code that belongs to speed inside the Van class., which makes it possible to reuse the Speed class in multiple applications.

In Object Oriented feature the users do not need to bother about which object is doing the real work. To achieve this, the Van class hides the implementation details from the users of the Van class. SO basically what happens is the users would ask the Van class to do a certain action and the Vann class will either do the work by itself or ask another class to perform the action.

9. A very important fact to remember is that Java only supports only single inheritance. This means that a class cannot extend more than one class. Therefore following is illegal:

```

public class extends Animal, Mammal{}

```

However a class can implement one or more interfaces. This has made Java get rid of the impossibility of multiple inheritance

10. Create a class Plants that has child classes of Trees and Shrubs. Create a Sub class of Trees as Fruit bearing tress. Identify attributes and methods of each class. Instantiate the fruit bearing tree class and show its attributes. Show the classes in the form of a tree as well (7)

**Web Resources:**

[http://www.tutorialspoint.com/java/java\\_inheritance.htm](http://www.tutorialspoint.com/java/java_inheritance.htm)

**Video Resource:**

<http://www.learnerstv.com/Free-Computers-Video-lectures-ltv006-Page1.htm>

**Summary:** This experiment allows the students to understand the concepts behind inheritance both as IS-A and HAS-A relations. The students are required to identify classes that follow inheritance.



## EXPERIMENT 6 – OVERLOADING

1. **Objectives:**
  - (a). Concept of overloading
2. **Time Required:** 3 hrs
3. **Programming Language:** Java
4. **Software Required:**
  - (a). Windows OS
  - (b). NetBeans / Eclipse Kepler
5. **Overriding:** Overloading allows different methods to have the same name, but different signatures where the signature can differ by the number of input parameters or type of input parameters or both. Overloading is related to compile time (or static) polymorphism. (1)

// Java program to demonstrate working of method  
// overloading in Java.

```
public class Sum {  
  
    // Overloaded sum(). This sum takes two int parameters  
    public int sum(int x, int y)  
    {  
        return (x + y);  
    }  
  
    // Overloaded sum(). This sum takes three int parameters  
    public int sum(int x, int y, int z)  
    {  
        return (x + y + z);  
    }  
  
    // Overloaded sum(). This sum takes two double parameters  
    public double sum(double x, double y)  
    {  
        return (x + y);  
    }  
    // Driver code  
    public static void main(String args[])  
    {  
        Sum s = new Sum();  
        System.out.println(s.sum(10, 20));  
        System.out.println(s.sum(10, 20, 30));  
        System.out.println(s.sum(10.5, 20.5));  
    }  
}
```

- 6      Execute the above code and show output. (3)
- 7      Can we overload methods on return type? (3)
- 8      Can we overload main() in Java? (3)

## EXPERIMENT 7 – OVERRIDING

1. **Objectives:**
  - (b). Concept of overriding
  - (c). Same method different usage
2. **Time Required:** 3 hrs
3. **Programming Language:** Java
4. **Software Required:**
  - (c). Windows OS
  - (d). NetBeans / Eclipse Kepler
5. **Overriding:** If a class inherits a method from its super class, then there is a (1) chance to override the method provided that it is not marked final. The benefit of overriding is: ability to define a behavior that's specific to the sub class type. Which means a subclass can implement a parent class method based on its requirement. In object oriented terms, overriding means to override the functionality of any existing method. Execute the following code and show the output. Discuss what is being overridden in this example?

```
class Animal{  
    public void move(){  
        System.out.println("Animals can move");  
    }  
}  
  
class Dog extends Animal{  
    public void move(){  
        System.out.println("Dogs can walk and run");  
    }  
}  
  
public class TestDog{  
    public static void main(String args[]){  
        Animal a = new Animal(); // Animal reference and object  
        Animal b = new Dog(); // Animal reference but Dog object  
  
        a.move();// runs the method in Animal class  
  
        b.move();//Runs the method in Dog class  
    }  
}
```

6. In the above example you can see that the even though **b** is a type of Animal it runs the move method in the Dog class. The reason for this is: In compile time the check is made on the reference type. However in the runtime JVM figures out the object type and would run the method that belongs to that particular object. Therefore in the above example, the program will compile properly since Animal class has the method move. Then at the runtime it runs the method specific for that object.
7. Execute the following and show output. What happened in the program and (1) Why?

```
class Animal{  
    public void move(){  
        System.out.println("Animals can move");  
    }  
}  
  
class Dog extends Animal{  
    public void move(){  
        System.out.println("Dogs can walk and run");  
    }  
    public void bark(){  
        System.out.println("Dogs can bark");  
    }  
}  
  
public class TestDog{  
    public static void main(String args[]){  
        Animal a = new Animal(); // Animal reference and object  
        Animal b = new Dog(); // Animal reference but Dog object  
  
        a.move();// runs the method in Animal class  
        b.move();//Runs the method in Dog class  
        b.bark();  
    }  
}
```



8. This program will throw a compile time error since b's reference type Animal doesn't have a method by the name of bark.

9. **Rules for Overriding**

- (a). The argument list should be exactly the same as that of the overridden method.
  - (b). The return type should be the same or a subtype of the return type declared in the original overridden method in the super class.
  - (c). The access level cannot be more restrictive than the overridden method's access level. For example: if the super class method is declared public then the overriding method in the sub class cannot be either private or protected.
  - (d). Instance methods can be overridden only if they are inherited by the subclass.
  - (e). A method declared final cannot be overridden.
  - (f). A method declared static cannot be overridden but can be re-declared.
  - (g). If a method cannot be inherited then it cannot be overridden.
  - (h). A subclass within the same package as the instance's superclass can override any superclass method that is not declared private or final.
  - (i). A subclass in a different package can only override the non-final methods declared public or protected.
  - (j). An overriding method can throw any unchecked exceptions, regardless of whether the overridden method throws exceptions or not. However the overriding method should not throw checked exceptions that are new or broader than the ones declared by the overridden method. The overriding method can throw narrower or fewer exceptions than the overridden method.
  - (k). Constructors cannot be overridden.
10. **Super Keyword:** When invoking a superclass version of an overridden method (1) the **super** keyword is used. Execute the following code and show output. What happened in this code?

```
class Animal{

    public void move(){
        System.out.println("Animals can move");
    }
}

class Dog extends Animal{

    public void move(){
        super.move(); // invokes the super class method
        System.out.println("Dogs can walk and run");
    }
}

public class TestDog{

    public static void main(String args[]){

        Animal b = new Dog(); // Animal reference but Dog object
        b.move(); //Runs the method in Dog class

    }
}
```

11. In the previous experiment create overridden method and call the method of a parent class using the child class. Show code and output (2)
12. Repeat Point 11 with calling the overridden method of the grandparent class. Show code and output. (2)

13. What are the merits and demerits of Overriding?

(3)

**Web Resources:**

[http://www.tutorialspoint.com/java/java\\_overriding.htm](http://www.tutorialspoint.com/java/java_overriding.htm)

**Video Lectures:**

<http://www.learnerstv.com/Free-Computers-Video-lectures-Itv006-Page1.htm>

**Summary:** This lab is about method overriding in Java. Also use of super keyword is also highlighted as used in Java.

## EXPERIMENT 8 – ABSTRACTION

### 1 **Objectives:**

- . (a). Abstraction
- . (b). Abstraction in Java
- . (c). Abstract methods

### 2 **Time Required:** 3 hrs

### 3 **Programming Language:** Java

### 4 **Software Required:**

- . (a). Windows OS
- . (b). NetBeans / Eclipse Kepler

5 **Abstraction:** Abstraction refers to the ability to make a class abstract in OOP. An abstract class is one that cannot be instantiated. All other functionality of the class still exists, and its fields, methods, and constructors are all accessed in the same manner. You just cannot create an instance of the abstract class.

If a class is abstract and cannot be instantiated, the class does not have much use unless it is subclassed. This is typically how abstract classes come about during the design phase. A parent class contains the common functionality of a collection of child classes, but the parent class itself is too abstract to be used on its own.

Use the **abstract** keyword to declare a class abstract. The keyword appears in the class declaration somewhere before the class keyword.

**/\* File name : Employee.java \*/**

```
public abstract 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;
    }
    public double computePay()
    {
        System.out.println("Inside Employee computePay");
        return 0.0;
    }
    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)
    {
        address = newAddress;
    }
    public int getNumber()
    {
        return number;
    }
}

```

Notice that nothing is different in this Employee class. The class is now abstract, but it still has three fields, seven methods, and one constructor.

- 6 Instantiate the class in point 5 and show outcome here? What happened? (1)

.

- 7 Abstract classes are extended in the following way (1)

.

```

/* File name : Salary.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()
    {
        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());
        return salary/52;
    }
}

```

```

/* File name : AbstractDemo.java */
public class AbstractDemo
{
    public static void main(String [] args)
    {
        Salary s = new Salary("Mohd Mohtashim", "Ambehta, UP", 3, 3600.00);
        Salary e = new Salary("John Adams", "Boston, MA", 2, 2400.00);

        System.out.println("Call mailCheck using Salary reference --");
        s.mailCheck();

        System.out.println("\n Call mailCheck using Employee reference--");
        e.mailCheck();
    }
}

```

Here we cannot instantiate a new Employee, but if we instantiate a new Salary object, the Salary object will inherit the three fields and seven methods from Employee. Execute and Show output here. What happened in this program?

- 8 **Abstract Methods:** If you want a class to contain a particular method but you want the actual implementation of that method to be determined by child classes, you can declare the method in the parent class as abstract. The abstract keyword is also used to declare a method as abstract. An abstract methods consist of a method signature, but no method body. Abstract method would have no definition, and its signature is followed by a semicolon, not curly braces as follows:

```
public abstract class Employee
{
    private String name;
    private String address;
    private int number;

    public abstract double computePay();

    //Remainder of class definition
}
```

Declaring a method as abstract has two results:

- (a). The class must also be declared abstract. If a class contains an abstract method, the class must be abstract as well.
- (b). Any child class must either override the abstract method or declare itself abstract.

A child class that inherits an abstract method must override it. If they do not, they must be abstract and any of their children must override it. Eventually, a descendant class has to implement the abstract method; otherwise, you would have a hierarchy of abstract classes that cannot be instantiated. If Salary is extending Employee class then it is required to implement computePay() method as follows:

```
/* File name : Salary.java */
public class Salary extends Employee
{
    private double salary; // Annual salary

    public double computePay()
    {
        System.out.println("Computing salary pay for " + getName());
        return salary/52;
    }

    //Remainder of class definition
}
```

- 9 Create an Abstract class student and create abstract methods. Create a class Fee to  
.  
invoke the abstract class. Show code and output here. Show the classes and methods  
graphically as well. (8)

**Web Resources:**

[http://www.tutorialspoint.com/java/java\\_abstraction.htm](http://www.tutorialspoint.com/java/java_abstraction.htm)

**Video Lectures:**

<http://www.learnerstv.com/Free-Computers-Video-lectures-ltv006-Page1.htm>

**Summary:** This lab session is with abstract classes and abstract methods in Java.  
Practice questions are also given.



## EXPERIMENT 9 – ENCAPSULATION

1 **Objectives:**

- . (a). Encapsulation
- . (b). Encapsulation in Java

2 **Time Required:** 3 hrs

3 **Programming Language:** Java

4 **Software Required:**

- . (a). Windows OS
- . (b). NetBeans / Eclipse Kepler

5 **Encapsulation** is one of the four fundamental OOP concepts. The other three are inheritance, polymorphism, and abstraction. Encapsulation is the technique of making the fields in a class private and providing access to the fields via public methods. If a field is declared private, it cannot be accessed by anyone outside the class, thereby hiding the fields within the class. For this reason, encapsulation is also referred to as data hiding.

Encapsulation can be described as a protective barrier that prevents the code and data being randomly accessed by other code defined outside the class. Access to the data and code is tightly controlled by an interface.

The main benefit of encapsulation is the ability to modify our implemented code without breaking the code of others who use our code. With this feature Encapsulation gives maintainability, flexibility and extensibility to our code.

6 The public methods are the access points to this class's fields from the outside java world. Normally these methods are referred as getters and setters. Therefore any class that wants to access the variables should access them through these getters and setters.

```

/* File name : EncapTest.java */
public class EncapTest{

    private String name;
    private String idNum;
    private int age;

    public int getAge(){
        return age;
    }

    public String getName(){
        return name;
    }

    public String getIdNum(){
        return idNum;
    }

    public void setAge( int newAge){
        age = newAge;
    }

    public void setName(String newName){
        name = newName;
    }

    public void setIdNum( String newId){
        idNum = newId;
    }
}

```

7 The variables of the EncapTest class can be access as below:

```

/* File name : RunEncap.java */
public class RunEncap{

    public static void main(String args[]){
        EncapTest encap = new EncapTest();
        encap.setName("James");
        encap.setAge(20);
        encap.setIdNum("12343ms");

        System.out.print("Name : " + encap.getName()+
                        " Age : "+ encap.getAge());
    }
}

```

Show the output here. What happened in the code? (8)

8 **Benefits:**

- . (a). The fields of a class can be made read-only or write-only.
  - (b). A class can have total control over what is stored in its fields.
  - (c). The users of a class do not know how the class stores its data. A class can change the data type of a field, and users of the class do not need to change any of their code.
- 9 For different classes maintaining inheritance create private and public methods/attributes. Show how and where they can be accessed using pen and pencil. (You may use the plants hierarchy for this task). (9)

**Web References:**

[http://www.tutorialspoint.com/java/java\\_encapsulation.htm](http://www.tutorialspoint.com/java/java_encapsulation.htm)

**Video Lectures:**

<http://www.learnerstv.com/Free-Computers-Video-lectures-ltv006-Page1.htm>

**Summary:** This lab introduces encapsulation concepts. Practice question provide a mechanism to implement encapsulation.

## EXPERIMENT 10 – INTERFACES

1. **Objectives:**
  - (a). Interfaces in Java
  - (b). Creating and using interfaces
2. **Time Required:** 3 hrs
3. **Programming Language:** Java
4. **Software Required:**
  - (a). Windows OS
  - (b). NetBeans / Eclipse Kepler
5. **Interface:** An interface is a collection of abstract methods. A class implements an interface, thereby inheriting the abstract methods of the interface. An interface is not a class. Writing an interface is similar to writing a class, but they are two different concepts. A class describes the attributes and behaviors of an object. An interface contains behaviors that a class implements.  
Unless the class that implements the interface is abstract, all the methods of the interface need to be defined in the class.  
An interface is similar to a class in the following ways:
  - (a). An interface can contain any number of methods.
  - (b). An interface is written in a file with a **.java** extension, with the name of the interface matching the name of the file. The bytecode of an interface appears in a **.class** file.
  - (c). Interfaces appear in packages, and their corresponding bytecode file must be in a directory structure that matches the package name.However, an interface is different from a class in several ways, including:
  - (a). You cannot instantiate an interface. An interface does not contain any constructors.
  - (b). All of the methods in an interface are abstract.
  - (c). An interface cannot contain instance fields. The only fields that can appear in an interface must be declared both static and final.
  - (d). An interface is not extended by a class; it is implemented by a class. An interface can extend multiple interfaces.
6. **Declaring Interfaces:** The **interface** keyword is used to declare an interface. Here is a simple example to declare an interface:

```
/* File name : NameOfInterface.java */
import java.lang.*;
//Any number of import statements

public interface NameOfInterface
{
    //Any number of final, static fields
    //Any number of abstract method declarations\
}
```

Interfaces have the following properties:

- (a). An interface is implicitly abstract. You do not need to use the **abstract** keyword when declaring an interface.
- (b). Each method in an interface is also implicitly abstract, so the abstract keyword is not needed. Methods in an interface are implicitly public.

```

/* File name : Animal.java */
interface Animal {

    public void eat();
    public void travel();
}

```

7. **Implementing Interfaces:** When a class implements an interface, you can think of the class as signing a contract, agreeing to perform the specific behaviors of the interface. If a class does not perform all the behaviors of the interface, the class must declare itself as abstract. Class uses the **implements** keyword to implement an interface. The implements keyword appears in the class declaration following the extends portion of the declaration.

```

/* File name : MammalInt.java */
public class MammalInt implements Animal{

    public void eat(){
        System.out.println("Mammal eats");
    }

    public void travel(){
        System.out.println("Mammal travels");
    }

    public int noOfLegs(){
        return 0;
    }

    public static void main(String args[]){
        MammalInt m = new MammalInt();
        m.eat();
        m.travel();
    }
}

```

Execute and Show output. What happened in this code? (1)

8. When overriding methods defined in interfaces there are several rules to be followed:
- (a). Checked exceptions should not be declared on implementation methods other than the ones declared by the interface method or subclasses of those declared by the interface method.
  - (b). The signature of the interface method and the same return type or subtype should be maintained when overriding the methods.
  - (c). An implementation class itself can be abstract and if so interface methods need not be implemented.

When implementation interfaces there are several rules:

- (a). A class can implement more than one interface at a time.
  - (b). A class can extend only one class, but implement many interface.
  - (c). An interface can extend another interface, similarly to the way that a class can extend another class.
9. **Extending Interfaces:** An interface can extend another interface, similarly to the way that a class can extend another class. The **extends** keyword is used to extend an interface, and the child interface inherits the methods of the parent interface. The following Sports interface is extended by Hockey and Football interfaces. The Hockey interface has four methods, but it inherits two from Sports; thus, a class that implements Hockey needs to implement all six methods. Similarly, a class that implements Football needs to define the three methods from Football and the two methods from Sports

```
//Filename: Sports.java
public interface Sports
{
    public void setHomeTeam(String name);
    public void setVisitingTeam(String name);
}

//Filename: Football.java
public interface Football extends Sports
{
    public void homeTeamScored(int points);
    public void visitingTeamScored(int points);
    public void endOfQuarter(int quarter);
}

//Filename: Hockey.java
public interface Hockey extends Sports
{
    public void homeGoalScored();
    public void visitingGoalScored();
    public void endOfPeriod(int period);
    public void overtimePeriod(int ot);
}
```

10. **Extending Multiple Interfaces:** A Java class can only extend one parent class. Multiple inheritances are not allowed. Interfaces are not classes, however, and an interface can extend more than one parent interface. The extends keyword is used once, and the parent interfaces are declared in a comma-separated list. For example, if the Hockey interface extended both Sports and Event, it would be declared as:

```
public interface Hockey extends Sports, Event
```

11. Create interfaces to the plants and trees hierarchy. Show code and show how the methods of an interface are used. Show output here as well (9)

**Web Resources:**

[http://www.tutorialspoint.com/java/java\\_interfaces.htm](http://www.tutorialspoint.com/java/java_interfaces.htm)

**Video Links:**

<http://www.learnerstv.com/Free-Computers-Video-lectures-ltv006-Page1.htm>

**Summary:** This lab identifies how interfaces are created used and extended. Interfaces allow a developer to create a child class having multiple parents.

## EXPERIMENT 11 – JAVA PACKAGES

1. **Objectives:**
  - (a). Packages
  - (b). Java and packages
2. **Time Required:** 3 hrs
3. **Programming Language:** Java
4. **Software Required:**
  - (a). Windows OS
  - (b). NetBeans / Eclipse Kepler
5. **Packages** are used in Java in-order to prevent naming conflicts, to control access, to make searching/locating and usage of classes, interfaces, enumerations and annotations easier etc.

A Package can be defined as a grouping of related types (classes, interfaces, enumerations and annotations) providing access protection and name space management.

Some of the existing packages in Java are::

- (a). **java.lang** - bundles the fundamental classes
- (b). **java.io** - classes for input , output functions are bundled in this package

Programmers can define their own packages to bundle group of classes/interfaces etc. It is a good practice to group related classes implemented by you so that a programmers can easily determine that the classes, interfaces, enumerations, annotations are related. Since the package creates a new namespace there won't be any name conflicts with names in other packages. Using packages, it is easier to provide access control and it is also easier to locate the related classed.

6. **Creating a Package:** When creating a package, you should choose a name for the package and put a **package** statement with that name at the top of every source file that contains the classes, interfaces, enumerations, and annotation types that you want to include in the package. The **package** statement should be the first line in the source file. There can be only one package statement in each source file, and it applies to all types in the file. If a package statement is not used then the class, interfaces, enumerations, and annotation types will be put into an unnamed package. Let us look at an example that creates a package called **animals**. It is common practice to use lowercased names of packages to avoid any conflicts with the names of classes, interfaces.

```
/* File name : Animal.java */
package animals;

interface Animal {
    public void eat();
    public void travel();
}
```



7.

```
package animals;

/* File name : MammalInt.java */
public class MammalInt implements Animal{

    public void eat(){
        System.out.println("Mammal eats");
    }

    public void travel(){
        System.out.println("Mammal travels");
    }

    public int noOfLegs(){
        return 0;
    }

    public static void main(String args[]){
        MammalInt m = new MammalInt();
        m.eat();
        m.travel();
    }
}
```

Execute the code and show output. What happened in this code? (1)

8. **Import Statements:** If a class wants to use another class in the same package, the package name does not need to be used. Classes in the same package find each other without any special syntax.

Here a class named `Boss` is added to the payroll package that already contains `Employee`. The `Boss` can then refer to the `Employee` class without using the payroll prefix, as demonstrated by the following `Boss` class.

```
package payroll;

public class Boss
{
    public void payEmployee(Employee e)
    {
        e.mailCheck();
    }
}
```

What happens if `Boss` is not in the payroll package? The `Boss` class must then use one of the following techniques for referring to a class in a different package.

- ▣ The fully qualified name of the class can be used. For example:

```
payroll.Employee
```

- ▣ The package can be imported using the import keyword and the wild card (\*). For example:

```
import payroll.*;
```

- ▣ The class itself can be imported using the import keyword. For example:

```
import payroll.Employee;
```

**Note:** A class file can contain any number of import statements. The import statements must appear after the package statement and before the class declaration.

9. Two major results occur when a class is placed in a package:
- (a). The name of the package becomes a part of the name of the class, as we just discussed in the previous section.
  - (b). The name of the package must match the directory structure where the corresponding bytecode resides.
10. Create a package that maintains the parent classes for plants and tress hierarchy.
- . Show code and outputs here (9)

**Web Resources:**

[http://www.tutorialspoint.com/java/java\\_packages.htm](http://www.tutorialspoint.com/java/java_packages.htm)

**Video Links:**

<http://www.learnerstv.com/Free-Computers-Video-lectures-Itv006-Page1.htm>

**Summary:** This lab works with creating and managing packages in Java. In creating a project a package may be required.

## EXPERIMENT 12 – POLYMORPHISM

1 **Objectives:**

- . (a). Polymorphism
- . (b). Polymorphism in Java

2 **Time Required:** 3 hrs

3 **Programming Language:** Java

4 **Software Required:**

- . (a). Windows OS
- . (b). NetBeans / Eclipse Kepler

5 **Polymorphism** is the ability of an object to take on many forms. The most common use of polymorphism in OOP occurs when a parent class reference is used to refer to a child class object. Any java object that can pass more than one IS-A test is considered to be polymorphic. In Java, all java objects are polymorphic since any object will pass the IS-A test for their own type and for the class Object. It is important to know that the only possible way to access an object is through a reference variable. A reference variable can be of only one type. Once declared the type of a reference variable cannot be changed. The reference variable can be reassigned to other objects provided that it is not declared final. The type of the reference variable would determine the methods that it can invoke on the object. A reference variable can refer to any object of its declared type or any subtype of its declared type. A reference variable can be declared as a class or interface type. Given the example

```
public interface Vegetarian{}  
public class Animal{}  
public class Deer extends Animal implements Vegetarian{}
```

Now the Deer class is considered to be polymorphic since this has multiple inheritance. Following are true for the above example:

- (a). A Deer IS-A Animal
- (b). A Deer IS-A Vegetarian
- (c). A Deer IS-A Deer
- (d). A Deer IS-A Object

When we apply the reference variable facts to a Deer object reference, the following declarations are legal:

```
Deer d = new Deer();  
Animal a = d;  
Vegetarian v = d;  
Object o = d;
```

- 6 **Virtual Methods:** As in 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.

```
/* File name : Employee.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;
    }
    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)
    {
        address = newAddress;
    }
    public int getNumber()
    {
        return number;
    }
}
```

Now if the extension is done as:

```
/* File name : Salary.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()
    {
        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());
        return salary/52;
    }
}
```

- 7 Execute the following and show the output. What happened in the program? Explain in detail (4)

```
/* File name : VirtualDemo.java */
public class VirtualDemo
{
    public static void main(String [] args)
    {
        Salary s = new Salary("Mohd Mohtashim", "Ambehta, UP", 3, 3600.00);
        Employee e = new Salary("John Adams", "Boston, MA", 2, 2400.00);
        System.out.println("Call mailCheck using Salary reference --");
        s.mailCheck();
        System.out.println("\n Call mailCheck using Employee reference--");
        e.mailCheck();
    }
}
```

- 8 Create polymorphs for the Tree and Plant hierarchy. Show code and outputs and highlight polymorphism in the codes. (6)

**Web Resources:**

[http://www.tutorialspoint.com/java/java\\_polymorphism.htm](http://www.tutorialspoint.com/java/java_polymorphism.htm)

**Video Links:**

<http://www.learnerstv.com/Free-Computers-Video-lectures-ltv006-Page1.htm>

**Summary:** This lab identifies how polymorphism is implemented in Java and its usage.

## EXPERIMENT 13 – GUI USING SWING IN JAVA

1. **Objectives:**
  - (a). GUI in Java
  - (b). Swing package
  - (c). Interacting with the user
2. **Time Required:** 3 hrs
3. **Programming Language:** Java
4. **Software Required:**
  - (a). Windows OS
  - (b). NetBeans / Eclipse Kepler

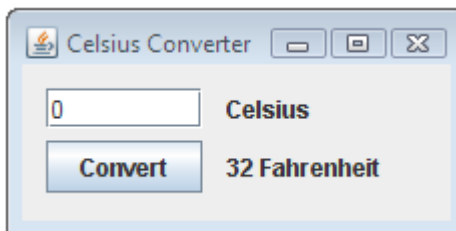
5. This lesson provides an introduction to Graphical User Interface (GUI) programming with Swing and the NetBeans IDE. As you learned in the "[Hello World!](#)" lesson, the NetBeans IDE is a free, open-source, cross-platform integrated development environment with built-in support for the Java programming language. It offers many advantages over coding with a text editor; we recommend its use whenever possible. If you have not yet read the above lesson, please take a moment to do so now. It provides valuable information about downloading and installing the JDK and NetBeans IDE.

The goal of this lesson is to introduce the Swing API by designing a simple application that converts temperature from Celsius to Fahrenheit. Its GUI will be basic, focusing on only a subset of the available Swing components. We will use the NetBeans IDE GUI builder, which makes user interface creation a simple matter of drag and drop. Its automatic code generation feature simplifies the GUI development process, letting you focus on the application logic instead of the underlying infrastructure.

Because this lesson is a step-by-step checklist of specific actions to take, we recommend that you run the NetBeans IDE and perform each step as you read along. This will be the quickest and easiest way to begin programming with Swing. If you are unable to do so, simply reading along should still be useful, since each step is illustrated with screenshots.

If you prefer the traditional approach of programming each component manually (without the assistance of an IDE), think of this lesson as an entry point into the lower-level discussions already provided elsewhere in the tutorial. Hyperlinks in each discussion will take you to related lessons, should you wish to learn such lower-level details.

The finished GUI for this application will look as follows:

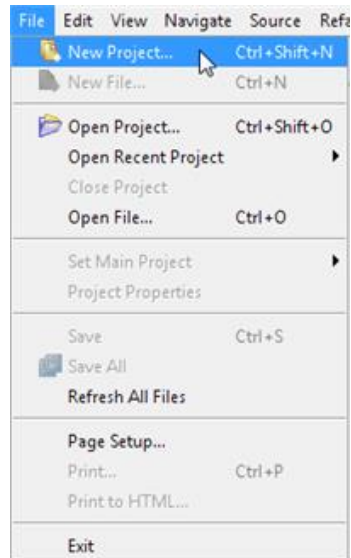


6. If you have worked with the NetBeans IDE in the past, much of this section will look familiar, since the initial steps are similar for most projects. Still, the following steps describe settings that are specific to this application, so take care to follow them closely.



## Step 1: Create a New Project

To create a new project, launch the NetBeans IDE and choose New Project from the File menu:

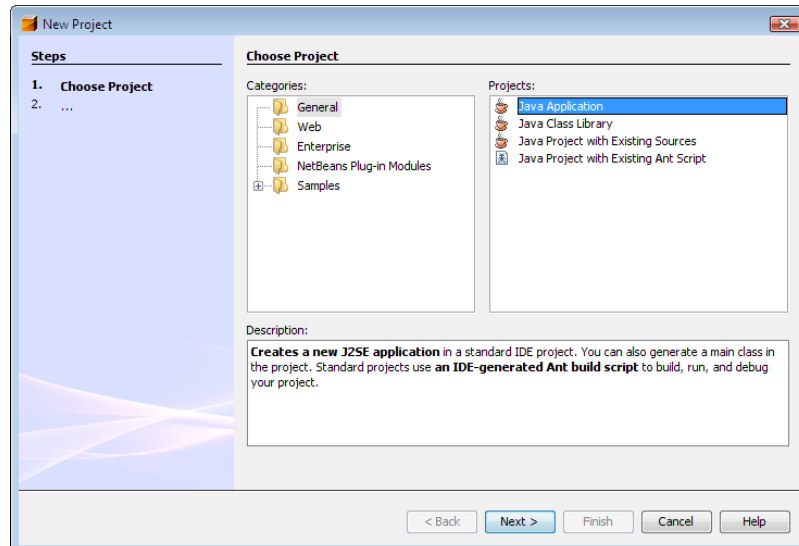


### Creating a New Project

Keyboard shortcuts for each command appear on the far right of each menu item. The look and feel of the NetBeans IDE may vary across platforms, but the functionality will remain the same.

## Step 2: Choose General -> Java Application

Next, select General from the Categories column, and Java Application from the Projects column:

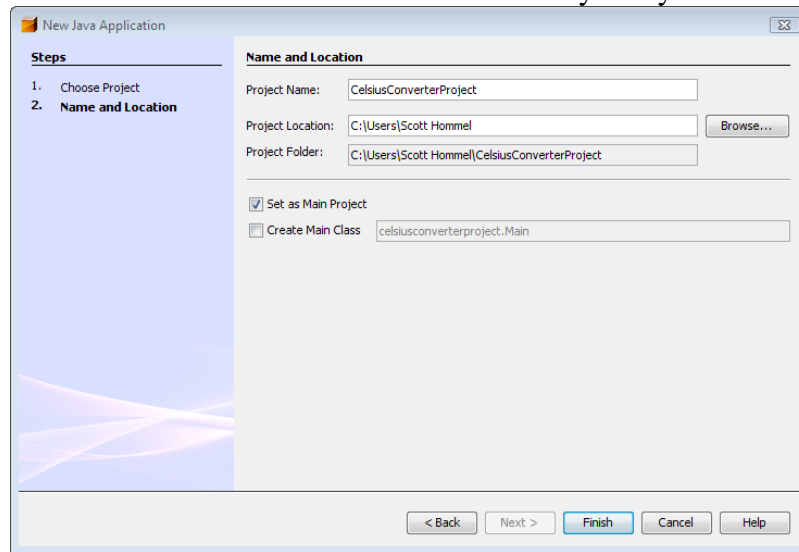


You may notice mention of "J2SE" in the description pane; that is the old name for what is now known as the "Java SE" platform. Press the button labelled "Next" to proceed.

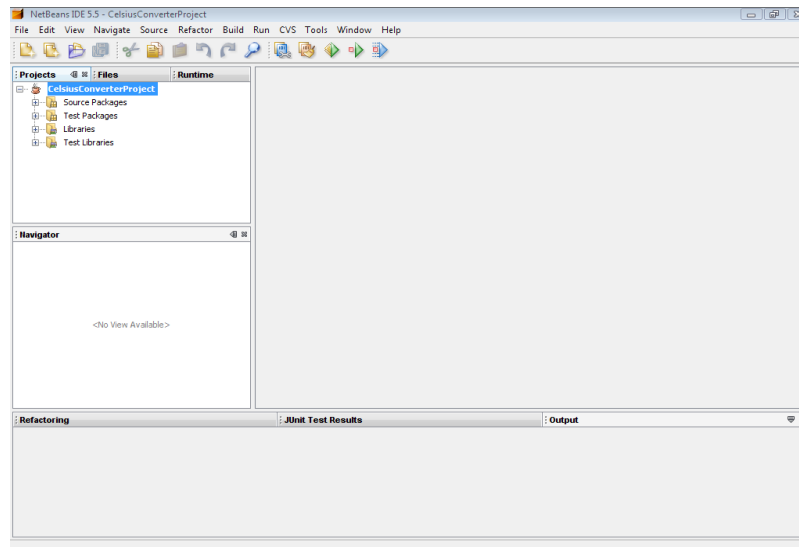
## Step 3: Set a Project Name

Now enter "CelsiusConverterProject" as the project name. You can leave the

Project Location and Project Folder fields set to their default values, or click the Browse button to choose an alternate location on your system.

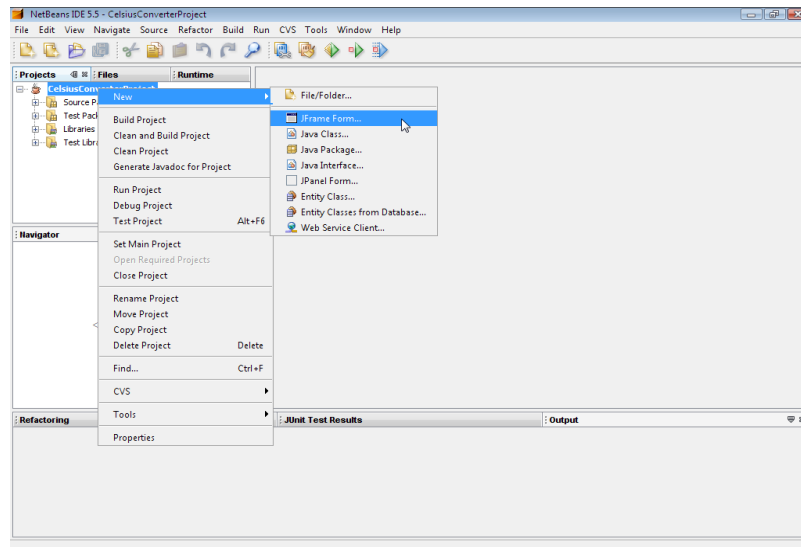


Make sure to deselect the "Create Main Class" checkbox; leaving this option selected generates a new class as the main entry point for the application, but our main GUI window (created in the next step) will serve that purpose, so checking this box is not necessary. Click the "Finish" button when you are done.



When the IDE finishes loading, you will see a screen similar to the above. All panes will be empty except for the Projects pane in the upper left hand corner, which shows the newly created project.

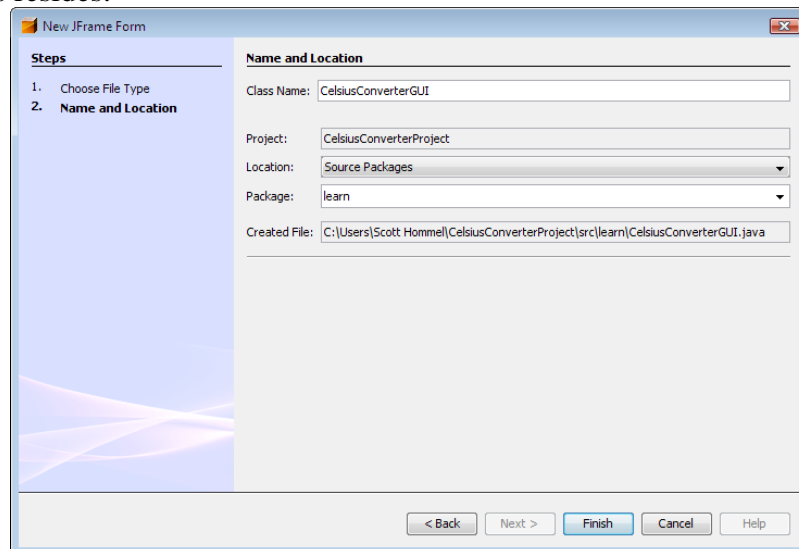
#### Step 4: Add a JFrame Form



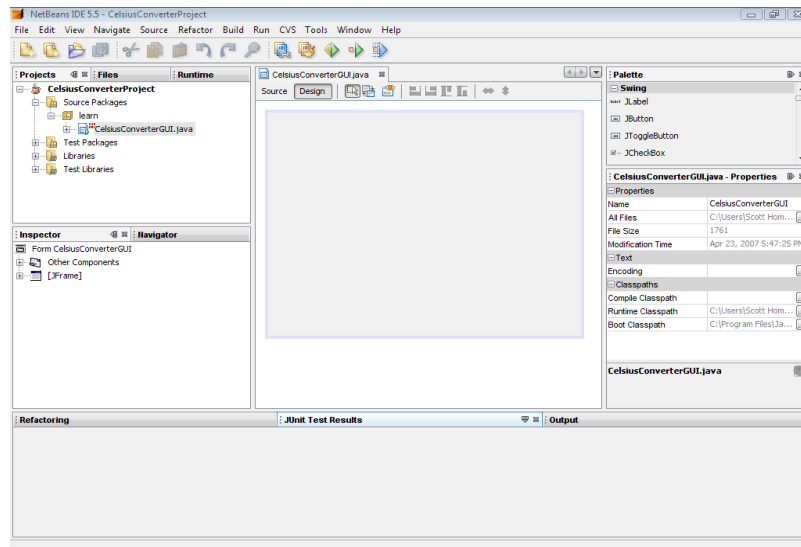
Now right-click the CelsiusConverterProject name and choose New -> JFrame Form (JFrame is the Swing class responsible for the main frame for your application.) You will learn how to designate this class as the application's entry point later in this lesson.

#### Step 5: Name the GUI Class

Next, type CelsiusConverterGUI as the class name, and learn as the package name. You can actually name this package anything you want, but here we are following the tutorial convention of naming the package after the lesson in which it resides.



The remainder of the fields should automatically be filled in, as shown above. Click the Finish button when you are done.

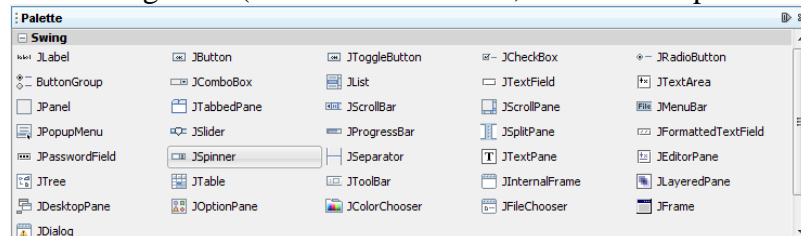


When the IDE finishes loading, the right pane will display a design-time, graphical view of the CelsiusConverterGUI. It is on this screen that you will visually drag, drop, and manipulate the various Swing components.

- It is not necessary to learn every feature of the NetBeans IDE before exploring its GUI creation capabilities. In fact, the only features that you really need to understand are the *Palette*, the *Design Area*, the *Property Editor*, and the *Inspector*. We will discuss these features below.

### The Palette

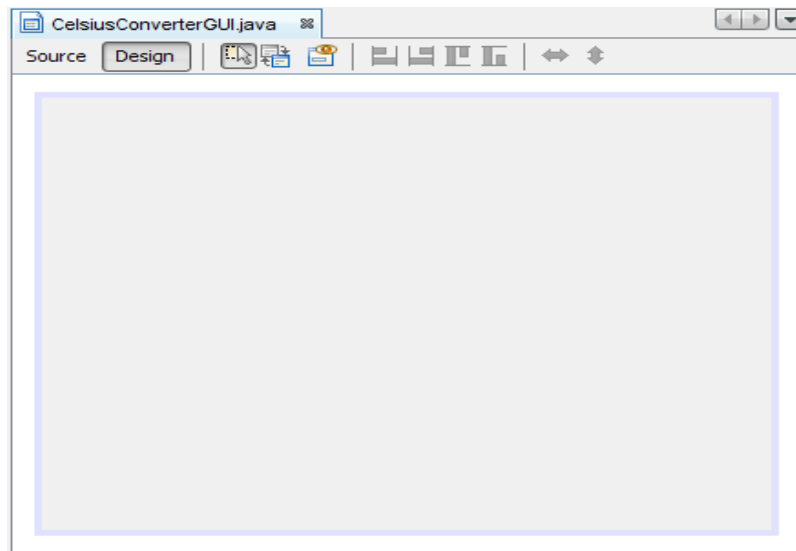
The Palette contains all of the components offered by the Swing API. You can probably already guess what many of these components are for, even if this is your first time using them (JLabel is a text label, JList is a drop-down list, etc.)



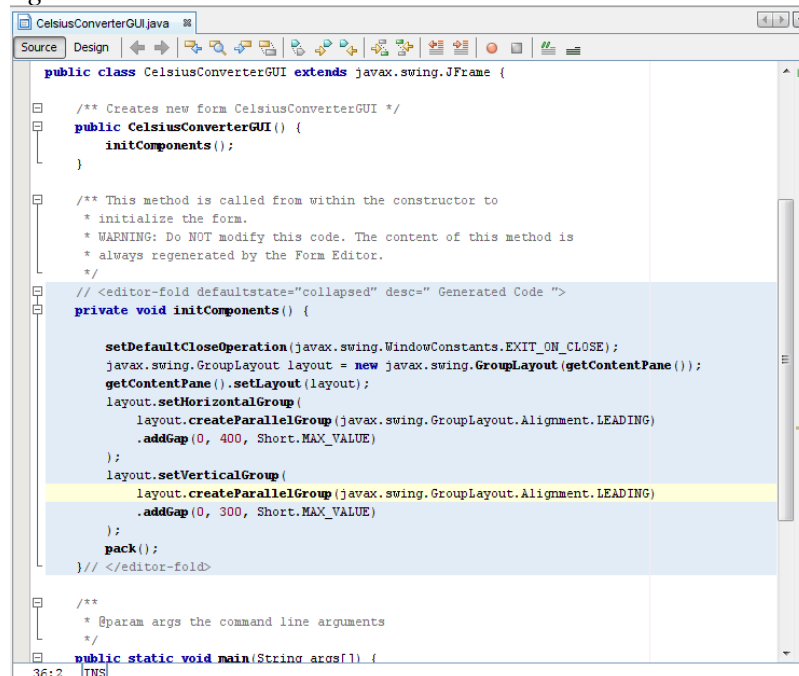
From this list, our application will use only JLabel (a basic text label), JTextField (for the user to enter the temperature), and JButton (to convert the temperature from Celsius to Fahrenheit.)

### The Design Area

The Design Area is where you will visually construct your GUI. It has two views: *source view*, and *design view*. Design view is the default, as shown below. You can toggle between views at any time by clicking their respective tabs.



The figure above shows a single JFrame object, as represented by the large shaded rectangle with blue border. Commonly expected behavior (such as quitting when the user clicks the "close" button) is auto-generated by the IDE and appears in the source view between un-editable blue sections of code known as *guarded blocks*.



A quick look at the source view reveals that the IDE has created a private method named `initComponents`, which initializes the various components of the GUI. It also tells the application to "exit on close", performs some layout-specific tasks, then packs the (soon to be added) components together on screen.

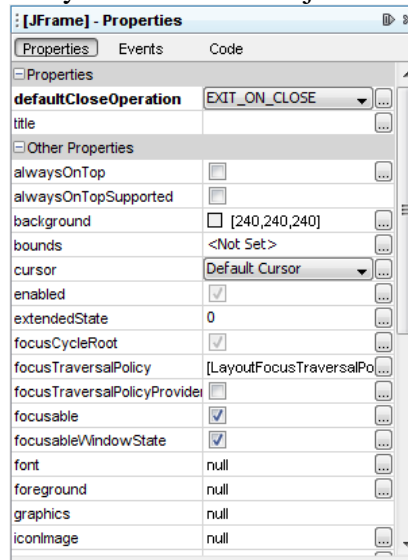
Don't feel that you need to understand this code in any detail; we mention it here simply to explore the source tab. For more information about these components, see:

[How to Make Frames \(Main Windows\)](#) and [Laying Out Components Within a](#)

## [Container.](#)

### The Property Editor

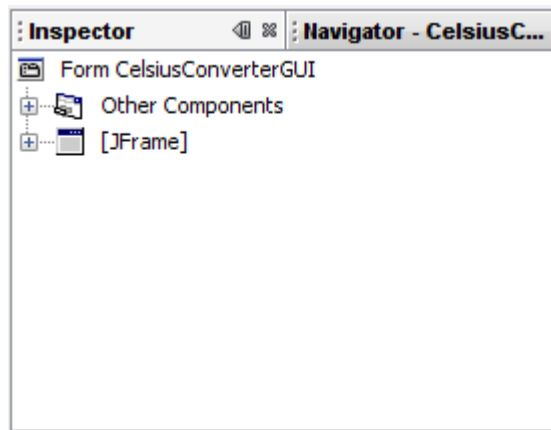
The Property Editor does what its name implies: it allows you to edit the properties of each component. The Property Editor is intuitive to use; in it you will see a series of rows — one row per property — that you can click and edit without entering the source code directly. The following figure shows the Property Editor for the newly added JFrame object:



The screenshot above shows the various properties of this object, such as background color, foreground color, font, and cursor.

### The Inspector

The last component of the NetBeans IDE that we will use in this lesson is the Inspector:



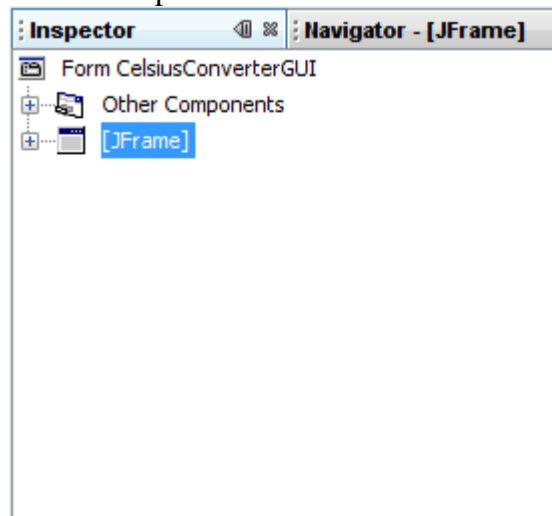
### The Inspector

The Inspector provides a graphical representation of your application's components. We will use the Inspector only once, to change a few variable names to something other than their defaults.

8. This section explains how to use the NetBeans IDE to create the application's GUI. As you drag each component from the Palette to the Design Area, the IDE auto-generates the appropriate source code.

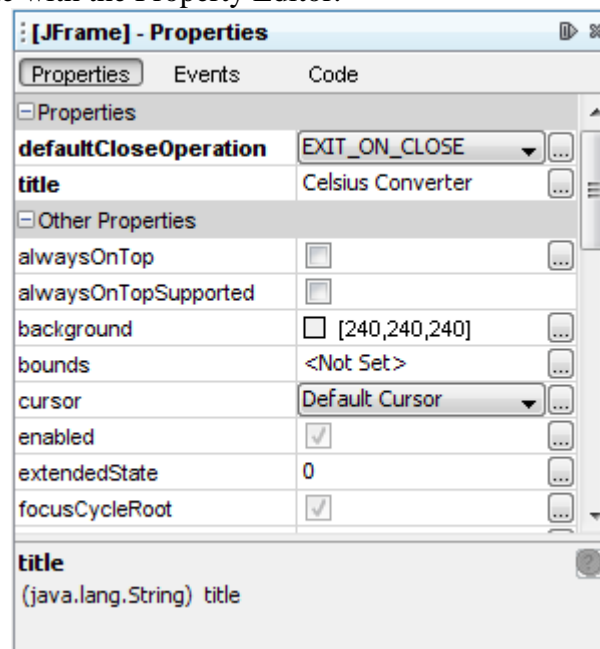
### Step 1: Set the Title

First, set the title of the application's JFrame to "Celsius Converter", by single-clicking the JFrame in the Inspector:




Selecting the JFrame

Then, set its title with the Property Editor:



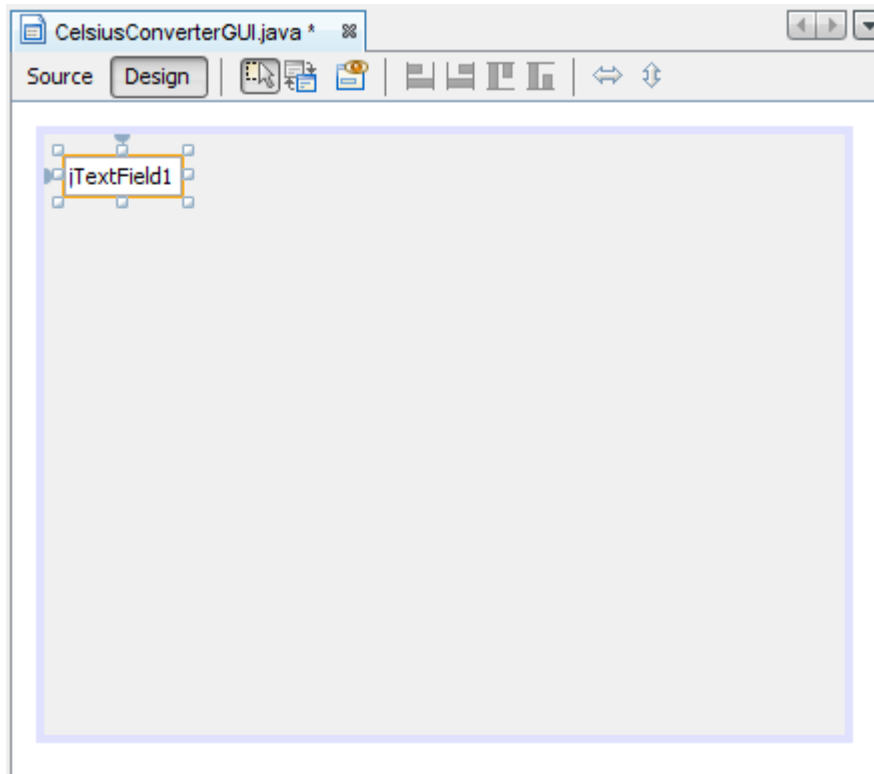
Setting the Title

You can set the title by either double-clicking the title property and entering the new text directly, or by clicking the  button and entering the title in the provided field. Or, as a shortcut, you could single-click the JFrame of the inspector and enter its new text directly without using the property editor.

### Step 2: Add a JTextField

Next, drag a JTextField from the Palette to the upper left corner of the Design Area. As you approach the upper left corner, the GUI builder provides visual cues (dashed lines) that suggest the appropriate spacing. Using these cues as a

guide, drop a JTextField into the upper left hand corner of the window as shown below:



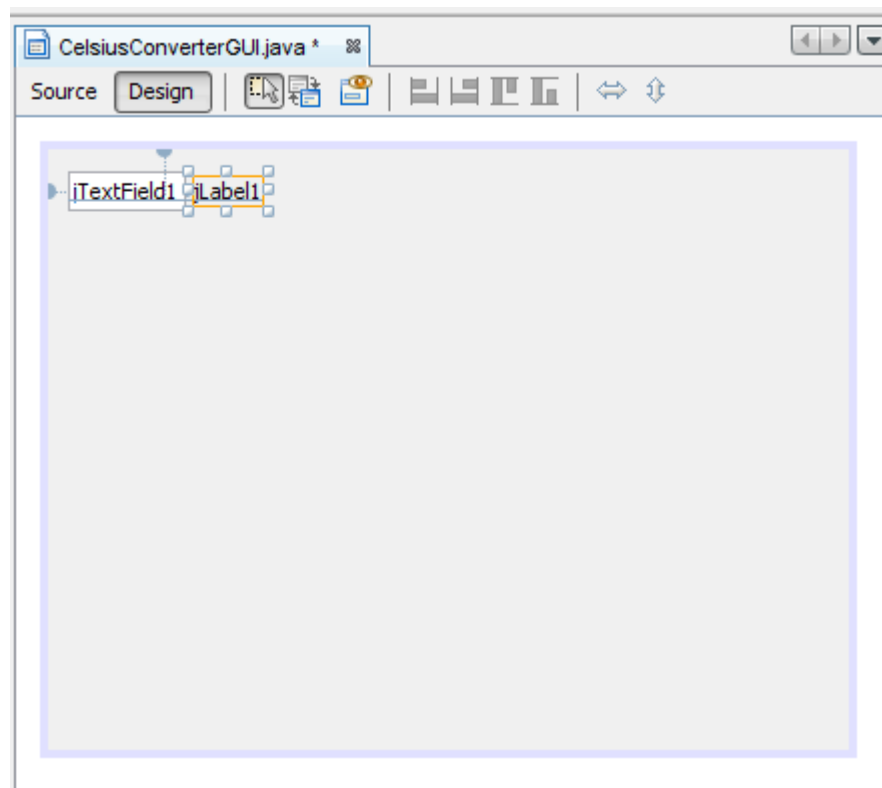
#### Adding a JTextField

You may be tempted to erase the default text "JTextField1", but just leave it in place for now. We will replace it later in this lesson as we make the final adjustments to each component. For more information about this component, see [How to Use Text Fields](#).

#### Step 3: Add a JLabel

Next, drag a JLabel onto the Design Area. Place it to the right of the JTextField, again watching for visual cues that suggest an appropriate amount of spacing. Make sure that text base for this component is aligned with that of the JTextField. The visual cues provided by the IDE should make this easy to determine.



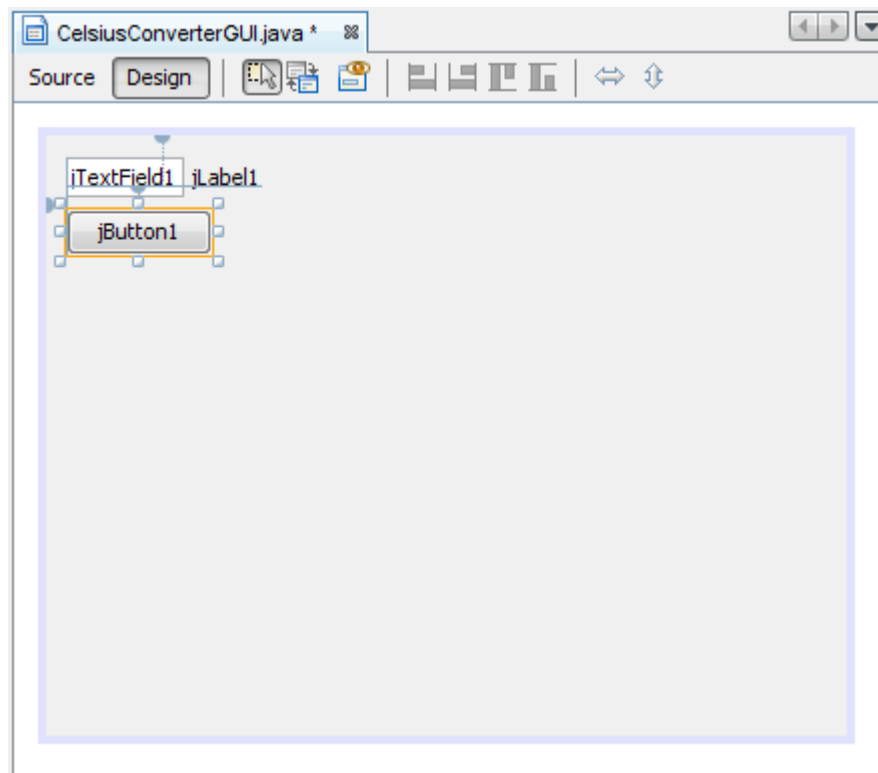


Adding a JLabel

For more information about this component, see [How to Use Labels](#).

#### **Step 4: Add a JButton**

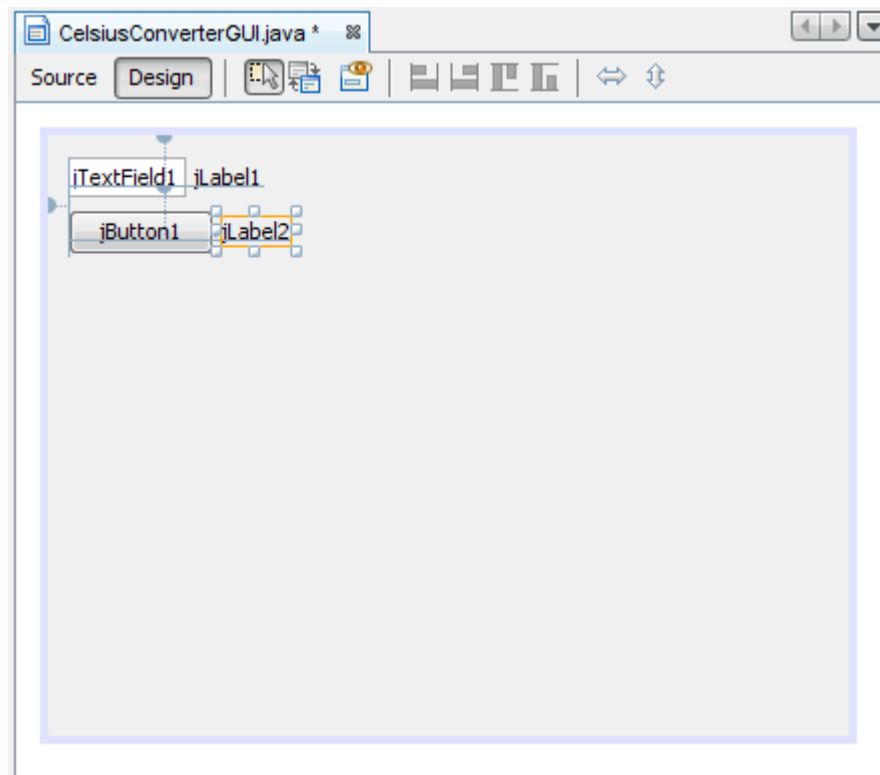
Next, drag a JButton from the Palette and position it to the left and underneath the JTextField. Again, the visual cues help guide it into place.



#### Adding a JButton

You may be tempted to manually adjust the width of the JButton and JTextField, but just leave them as they are for now. You will learn how to correctly adjust these components later in this lesson. For more information about this component, see [How to Use Buttons](#).

#### **Step 5: Add a Second JLabel**



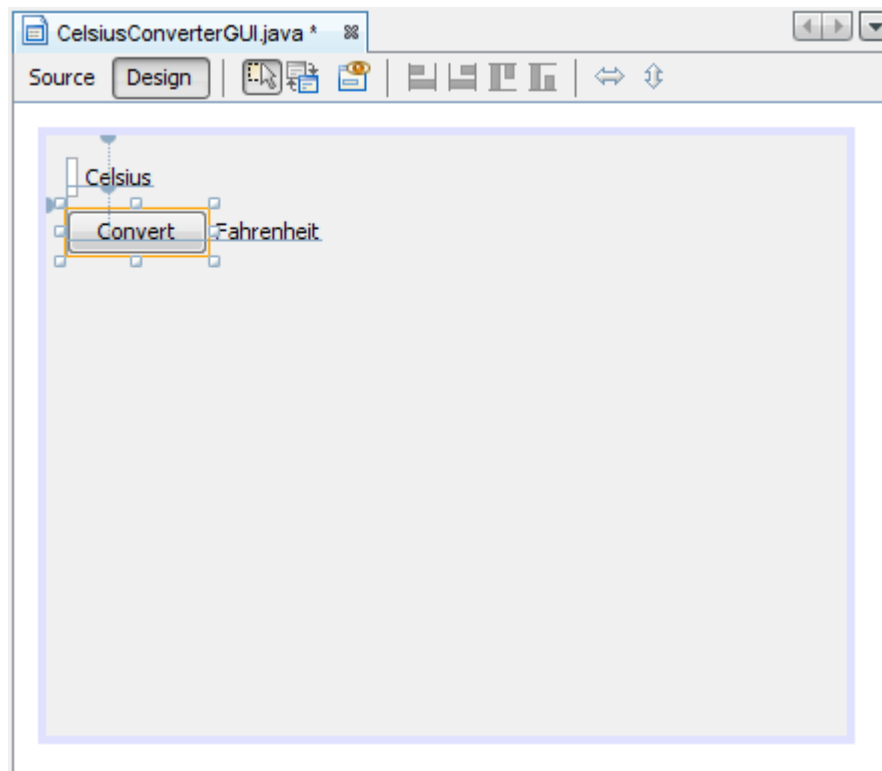
#### Adding a Second JLabel

Finally, add a second JLabel, repeating the process in step 2. Place this second label to the right of the JButton, as shown above.

9. With the GUI components now in place, it is time to make the final adjustments. There are a few different ways to do this; the order suggested here is just one possible approach.

#### **Step 1: Set the Component Text**

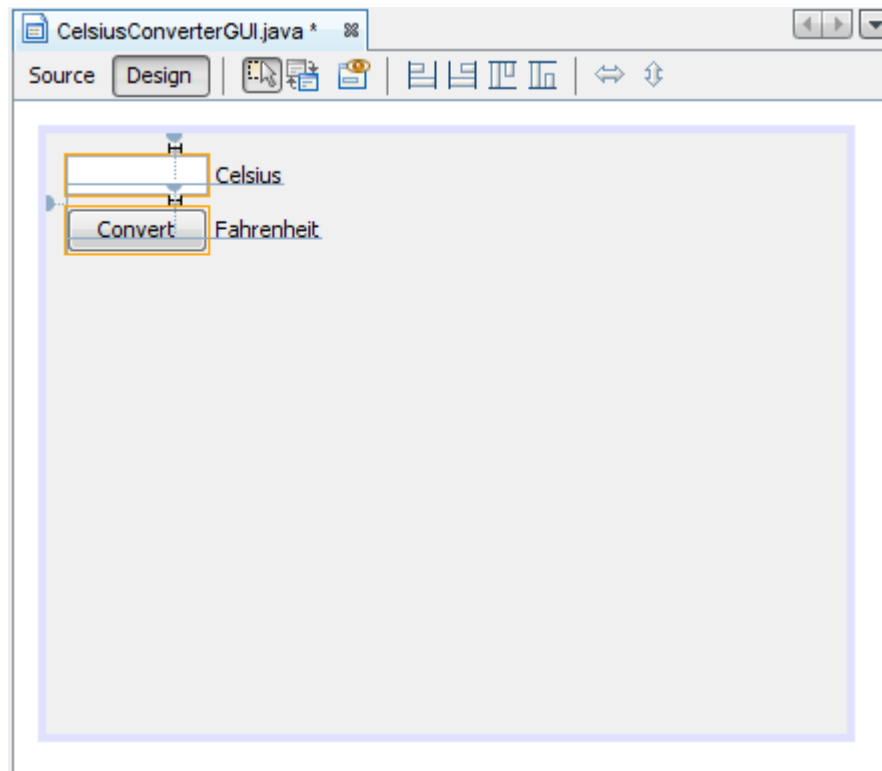
First, double-click the JTextField and JButton to change the default text that was inserted by the IDE. When you erase the text from the JTextField, it will shrink in size as shown below. Change the text of the JButton from "JButton1" to "Convert." Also change the top JLabel text to "Celsius" and the bottom to "Fahrenheit."



Setting the Component Text

**Step 2: Set the Component Size**

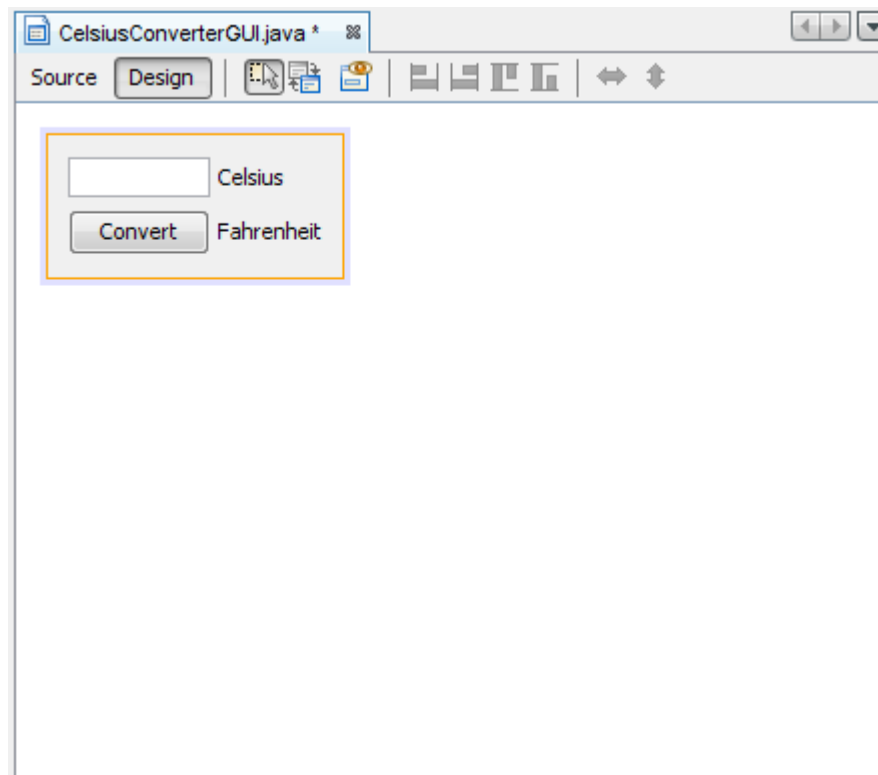
Next, shift-click the JTextField and JButton components. This will highlight each showing that they are selected. Right-click (control-click for mac users) Same Size -> Same Width. The components will now be the same width, as shown below. When you perform this step, make sure that JFrame itself is not also selected. If it is, the Same Size menu will not be active.



Setting the JTextField and JButton Sizes

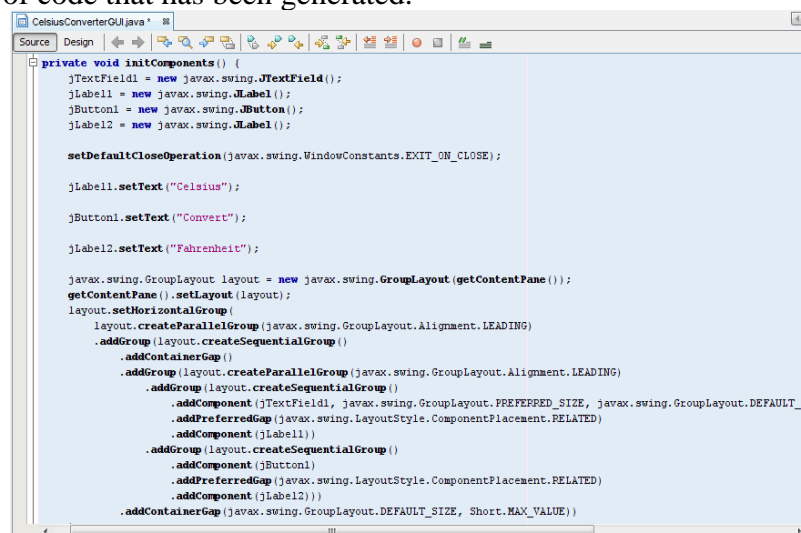
### **Step 3: Remove Extra Space**

Finally, grab the lower right-hand corner of the JFrame and adjust its size to eliminate any extra whitespace. Note that if you eliminate all of the extra space (as shown below) the title (which only appears at runtime) may not show completely. The end-user is free to resize the application as desired, but you may want to leave some extra space on the right side to make sure that everything fits correctly. Experiment, and use the screenshot of the finished GUI as a guide.



## The Completed GUI

The GUI portion of this application is now complete! If the NetBeans IDE has done its job, you should feel that creating this GUI was a simple, if not trivial, task. But take a minute to click on the source tab; you might be surprised at the amount of code that has been generated.



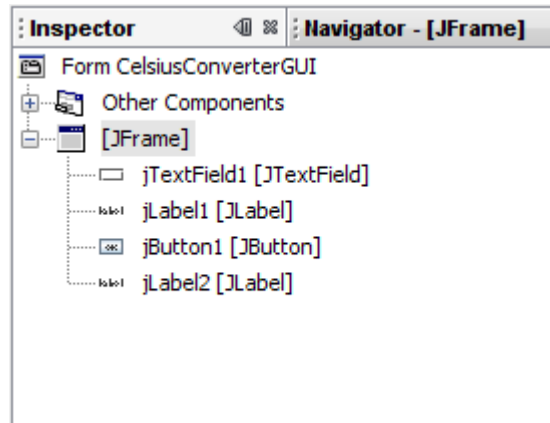
To see the code in its entirety, scroll up and down within the IDE as necessary. You can expand or collapse certain blocks of code (such as method bodies) by clicking the + or - symbol on the left-hand side of the source editor.

## 10. Adding the Application Logic

It is now time to add in the application logic.

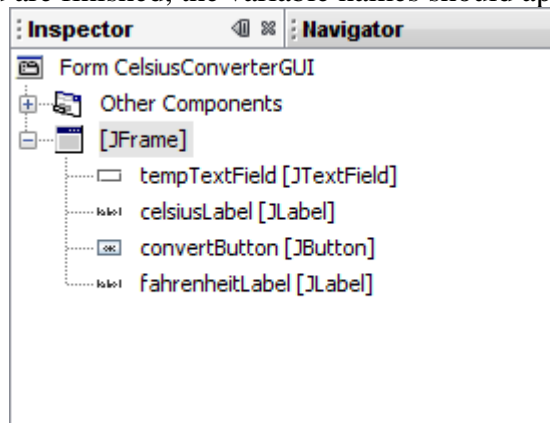
### Step 1: Change the Default Variable Names

The figure below shows the default variable names as they currently appear within the Inspector. For each component, the variable name appears first, followed by the object's type in square brackets. For example, jTextField1 [JTextField] means that "jTextField1" is the variable name and "JTextField" is its type.



Default Variable Names

The default names are not very relevant in the context of this application, so it makes sense to change them from their defaults to something that is more meaningful. Right-click each variable name and choose "Change variable name." When you are finished, the variable names should appear as follows:



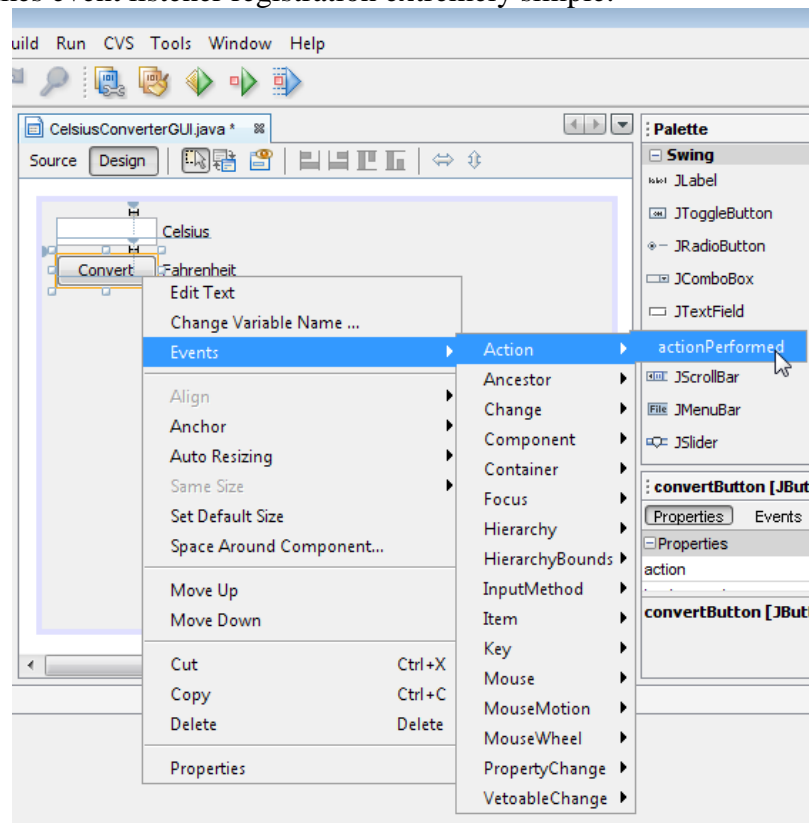
New Variable Names

The new variable names are "tempTextField", "celsiusLabel", "convertButton", and "fahrenheitLabel." Each change that you make in the Inspector will automatically propagate its way back into the source code. You can rest assured that compilation will not fail due to typos or mistakes of that nature — mistakes that are common when editing by hand.

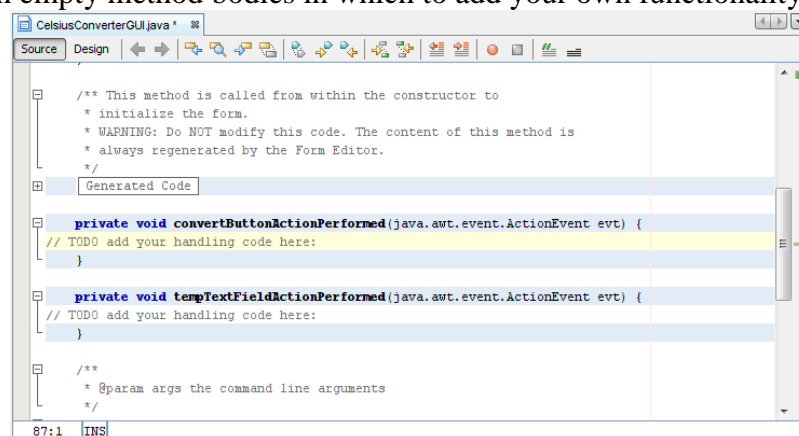
### Step 2: Register the Event Listeners

When an end-user interacts with a Swing GUI component (such as clicking the Convert button), that component will generate a special kind of object — called an *event object* — which it will then broadcast to any other objects that

have previously registered themselves as *listeners* for that event. The NetBeans IDE makes event listener registration extremely simple:



In the Design Area, click on the Convert button to select it. Make sure that *only* the Convert button is selected (if the JFrame itself is also selected, this step will not work.) Right-click the Convert button and choose Events -> Action -> ActionPerformed. This will generate the required event-handling code, leaving you with empty method bodies in which to add your own functionality:



There are many different event types representing the various kinds of actions that an end-user can take (clicking the mouse triggers one type of event, typing at the keyboard triggers another, moving the mouse yet another, and so on.) Our application is only concerned with the `ActionEvent`; for more information about event handling, see [Writing Event Listeners](#).



### Step 3: Add the Temperature Conversion Code

The final step is to simply paste the temperature conversion code into the empty method body. The following code is all that is necessary to convert a temperature from Celsius to Fahrenheit:

---

#### Note:

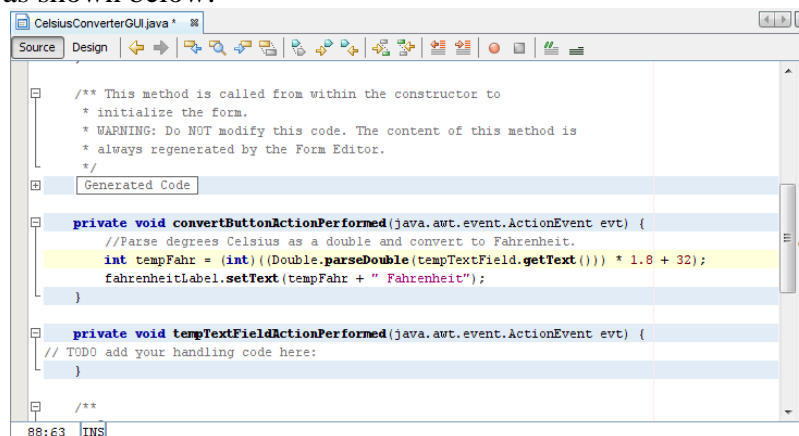
This example is not localizable because the `parseDouble` method is not localizable. This code snippet is for illustration purposes only. A more robust implementation would use the [Scanner](#) class to parse the user input.

---

```
//Parse degrees Celsius as a double and convert to Fahrenheit.  
int tempFahr = (int)((Double.parseDouble(tempTextField.getText()))  
    * 1.8 + 32);
```

```
fahrenheitLabel.setText(tempFahr + " Fahrenheit");
```

Simply copy this code and paste it into the `convertButtonActionPerformed` method as shown below:



With the conversion code in place, the application is now complete.

### Step 4: Run the Application

Running the application is simply a matter of choosing Run -> Run Main Project within the NetBeans IDE. The first time you run this application, you will be prompted with a dialog asking to set CelsiusConverterGUI as the main class for this project. Click the OK button, and when the program finishes compiling, you should see the application running in its own window.

11. Create a Swing based GUI application that takes as input two Cartesian points and returns the Euclidean distance among those points. Show the Output here and execute the code as a viva. (10)

**Web Resources:**

<http://docs.oracle.com/javase/tutorial/uiswing/learn/index.html>

**Video Links**

<http://www.youtube.com/watch?v=LFr06ZKIpSM>

<http://www.youtube.com/watch?v=w3CD8qxDMXk>

<http://www.youtube.com/watch?v=Ph8GoNlbwgg>

<http://www.youtube.com/watch?v=vQE85ndkFBY>

<http://www.youtube.com/watch?v=wi5hg3onjd0>

**Example Code**

<http://docs.oracle.com/javase/tutorial/uiswing/examples/learn/index.html>

**Summary:** This lab session introduces the GUI using Swing components by creating a simple application. An exercise is provided to help the students in creating their own application.

## EXPERIMENT 14 – EXCEPTION HANDLING

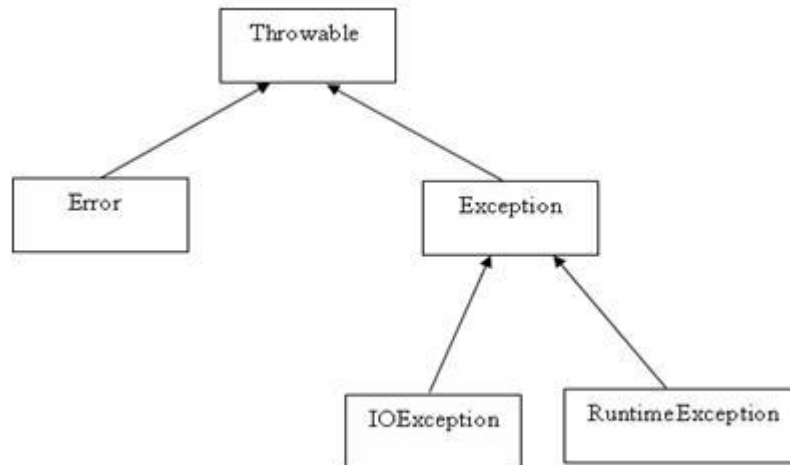
1. **Objectives:**
  - (a). Program exceptions
  - (b). Exception handling using Try Catch blocks
  - (c). Creating and handling own exceptions
2. **Time Required:** 3 hrs
3. **Programming Language:** Java
4. **Software Required:**
  - (a). Windows OS
  - (b). NetBeans / Eclipse Kepler
5. **Exceptions:** 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). A user has entered invalid data.
  - (b). A file that needs to be opened cannot be found.
  - (c). 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.
6. To understand how exception handling works in Java, you need to understand the three categories of exceptions:
  - (a). **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.
  - (b). **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.
  - (c). **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
7. **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 from 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.

Here is a list of most common checked and unchecked [Java's Built-in Exceptions](#).



8. **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:

```
try
{
    //Protected code
} catch (ExceptionName e1)
{
    //Catch block
}
```

A catch statement involves declaring the type of exception you are trying to catch. If an exception occurs in protected code, the catch block (or blocks) that follows the try is checked. If the type of exception that occurred is listed in a catch block, the exception is passed to the catch block much as an argument is passed into a method parameter.

The following is an array is declared with 2 elements. Then the code tries to access the 3rd element of the array which throws an exception.

```
// File Name : ExcepTest.java
import java.io.*;
public class ExcepTest{

    public static void main(String args[]){
        try{
            int a[] = new int[2];
            System.out.println("Access element three : " + a[3]);
        } catch (ArrayIndexOutOfBoundsException e) {
            System.out.println("Exception thrown : " + e);
        }
        System.out.println("Out of the block");
    }
}
```

This would produce the following result:

```
Exception thrown :java.lang.ArrayIndexOutOfBoundsException: 3
Out of the block
```

## 9. Multiple Exceptions

A try block can be followed by multiple catch blocks. The syntax for multiple catch blocks looks like the following:

```
try
{
    //Protected code
} catch (ExceptionType1 e1)
{
    //Catch block
} catch (ExceptionType2 e2)
{
    //Catch block
} catch (ExceptionType3 e3)
{
    //Catch block
}
```

The previous statements demonstrate three catch blocks, but you can have any number of them after a single try. If an exception occurs in the protected code, the exception is thrown to the first catch block in the list. If the data type of the exception thrown matches ExceptionType1, it gets caught there. If not, the exception passes down to the second catch statement. This continues until the exception either is caught or falls through all catches, in which case the current method stops execution and the exception is thrown down to the previous method on the call stack.

Here is code segment showing how to use multiple try/catch statements.

```
try
{
    file = new FileInputStream(fileName);
    x = (byte) file.read();
} catch (IOException i)
{
    i.printStackTrace();
    return -1;
} catch (FileNotFoundException f) //Not valid!
{
    f.printStackTrace();
    return -1;
}
```

## 10 The Throws Keyword:

If a method does not handle a checked exception, the method must declare it using the **throws** keyword. The throws keyword appears at the end of a method's signature. You can throw an exception, either a newly instantiated one or an exception that you just caught, by using the **throw** keyword. Try to understand the difference in throws and throw keywords.

The following method declares that it throws a RemoteException:

```
import java.io.*;
public class className
{
    public void deposit(double amount) throws RemoteException
    {
        // Method implementation
        throw new RemoteException();
    }
    //Remainder of class definition
}
```

A method can declare that it throws more than one exception, in which case the exceptions are declared in a list separated by commas. For example, the following method declares that it throws a RemoteException and an InsufficientFundsException:

```
import java.io.*;
public class className
{
    public void withdraw(double amount) throws RemoteException,
        InsufficientFundsException
    {
        // Method implementation
    }
    //Remainder of class definition
}
```

#### 11 **The finally Keyword:**

The finally keyword is used to create a block of code that follows a try block. A finally block of code always executes, whether or not an exception has occurred. Using a finally block allows you to run any cleanup-type statements that you want to execute, no matter what happens in the protected code. A finally block appears at the end of the catch blocks and has the following syntax:

```
try
{
    //Protected code
} catch(ExceptionType1 e1)
{
    //Catch block
} catch(ExceptionType2 e2)
{
    //Catch block
} catch(ExceptionType3 e3)
{
    //Catch block
} finally
{
    //The finally block always executes.
}
```

Here is the code

```

public class ExceptTest{

    public static void main(String args[]){
        int a[] = new int[2];
        try{
            System.out.println("Access element three :" + a[3]);
        }catch(ArrayIndexOutOfBoundsException e){
            System.out.println("Exception thrown  :" + e);
        }
        finally{
            a[0] = 6;
            System.out.println("First element value: " +a[0]);
            System.out.println("The finally statement is executed");
        }
    }
}

```

This would produce the following result:

```

Exception thrown :java.lang.ArrayIndexOutOfBoundsException: 3
First element value: 6
The finally statement is executed

```

## 12 **Declaring Your Own Exceptions:**

- . You can create your own exceptions in Java. Keep the following points in mind when writing your own exception classes:
  - (a). All exceptions must be a child of Throwable.
  - (b). If you want to write a checked exception that is automatically enforced by the Handle or Declare Rule, you need to extend the Exception class.
  - (c). If you want to write a runtime exception, you need to extend the RuntimeException class.

We can define our own Exception class as below:

```

class MyException extends Exception{
}

```

You just need to extend the Exception class to create your own Exception class. These are considered to be checked exceptions. The following InsufficientFundsException class is a user-defined exception that extends the Exception class, making it a checked exception. An exception class is like any other class, containing useful fields and methods.

```
// File Name InsufficientFundsException.java
import java.io.*;

public class InsufficientFundsException extends Exception
{
    private double amount;
    public InsufficientFundsException(double amount)
    {
        this.amount = amount;
    }
    public double getAmount()
    {
        return amount;
    }
}
```

To demonstrate using our user-defined exception, the following CheckingAccount class contains a withdraw() method that throws an InsufficientFundsException.

```
// File Name CheckingAccount.java
import java.io.*;

public class CheckingAccount
{
    private double balance;
    private int number;
    public CheckingAccount(int number)
    {
        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);
        }
    }
    public double getBalance()
    {
        return balance;
    }
    public int getNumber()
    {
        return number;
    }
}
```



The following BankDemo program demonstrates invoking the deposit() and withdraw() methods of CheckingAccount.

```
// File Name BankDemo.java
public class BankDemo
{
    public static void main(String [] args)
    {
        CheckingAccount c = new CheckingAccount(101);
        System.out.println("Depositing $500...");
        c.deposit(500.00);
        try
        {
            System.out.println("\nWithdrawing $100...");
            c.withdraw(100.00);
            System.out.println("\nWithdrawing $600...");
            c.withdraw(600.00);
        } catch (InsufficientFundsException e)
        {
            System.out.println("Sorry, but you are short $"
                               + e.getAmount());
            e.printStackTrace();
        }
    }
}
```

- 13 Compile the files given in Point 12 and execute BankDemo. Show the output here.  
Also Make a trace call using pencil and pen on the given code to show the execution.  
(3)

- 14 Identify at least 3 different exception classes for the plants and trees hierarchy.  
Provide try catch block as well as finally statement here. Execute the code and show output for exception behaviour. Also in your code trace the exceptional calls using pencil and show the code here. (7)

**Web Resources:**

[http://www.tutorialspoint.com/java/java\\_exceptions.htm](http://www.tutorialspoint.com/java/java_exceptions.htm)

[http://www.tutorialspoint.com/java/java\\_builtin\\_exceptions.htm](http://www.tutorialspoint.com/java/java_builtin_exceptions.htm)

**Video Links:**

<http://www.youtube.com/watch?v=4my7mKFaNQs>

<http://www.youtube.com/watch?v=LpfHEjEoDCg>

[http://www.youtube.com/watch?v=K\\_-3OLkXkzY](http://www.youtube.com/watch?v=K_-3OLkXkzY)

**Summary:** This lab identifies how exceptions are handled in Java. Exceptions are classes that allow the developers to identify alternate program flow. Students are given an opportunity to create their own exceptions and show them using traces.

## EXPERIMENT 15 – PAINT APPLICATION IN JAVA USING NETBEANS

1. **Objectives:**

- (a). Creating a Paint application in NetBeans
- (b). Creating and embedding canvas in a paint window
- (c). Running, branding and packaging the project

2. **Time Required:** 3 hrs

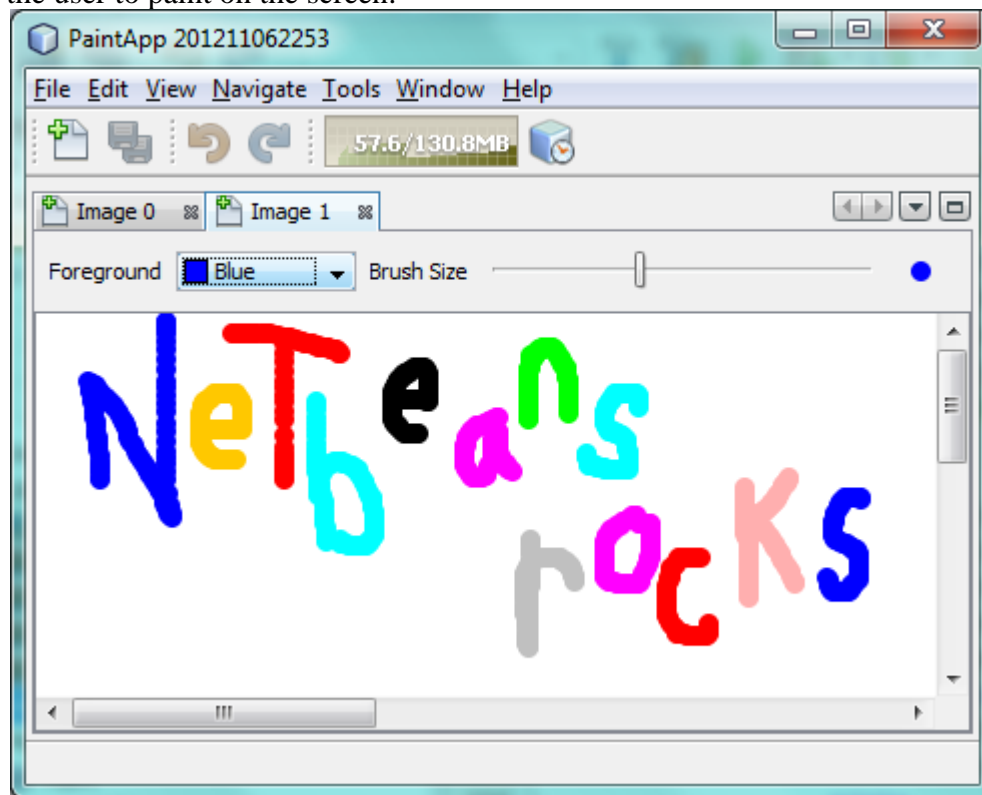
3. **Programming Language:** Java

4. **Software Required:**

- (a). Windows OS
- (b). NetBeans

5. **Paint Application:**

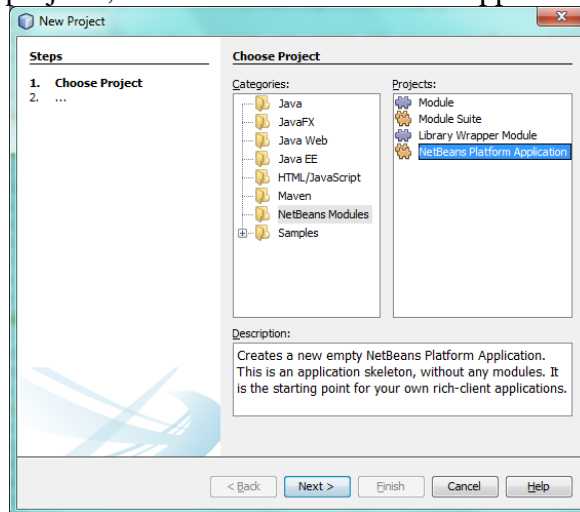
You will create a simple application on the NetBeans Platform. The application allows the user to paint on the screen:



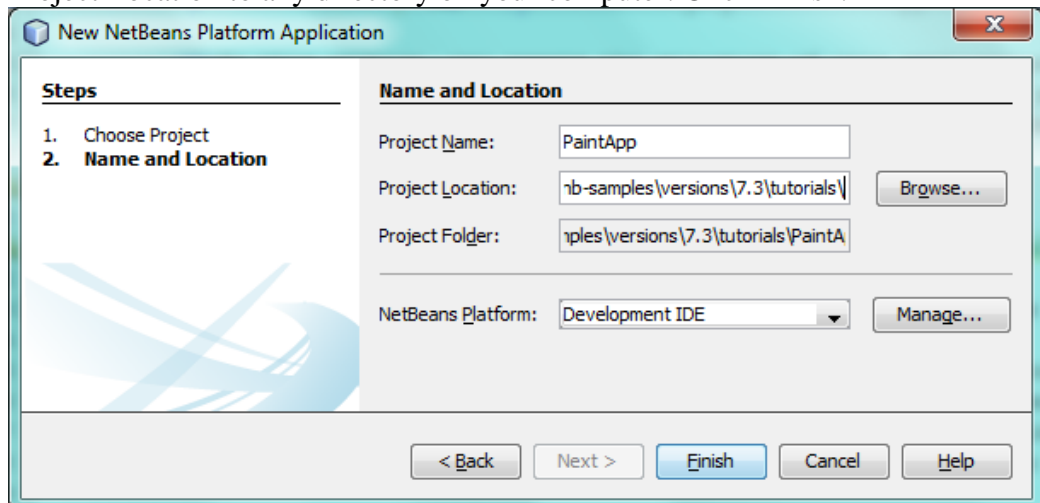
This initial version is far from a full-fledged paint application, but it demonstrates a very simple case of creating an application on top of the NetBeans Platform.

6. **Creating the Application Skeleton:** The "NetBeans Platform Application" template will create your application's skeleton. The skeleton will consist of a set of modules that work together to form the basis of your application. You will use the Project Properties dialog to assign your application's splashscreen, application name, and the type and number of NetBeans modules that you want to use. You can also take advantage of such actions as creating a ZIP distribution and building a Java WebStart (JNLP) application, which are important tools in making your application available to other users.

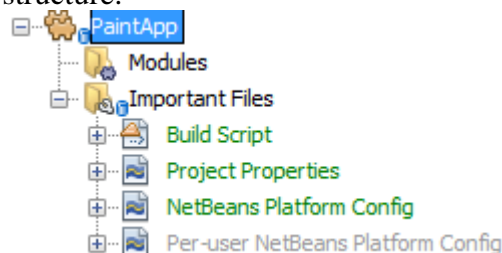
- (a). Choose File > New Project. Under Categories, select NetBeans Modules. Under projects, select NetBeans Platform Application: Click Next.



- (b). In the Name and Location panel, type PaintApp in Project Name. Change the Project Location to any directory on your computer. Click Finish.

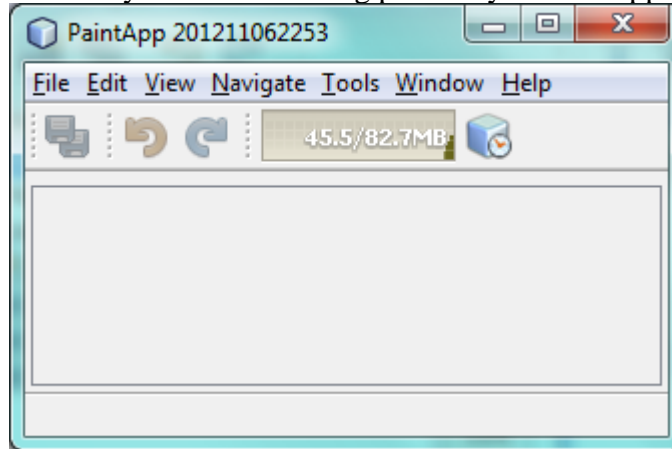


- (c). The new application skeleton opens in the IDE. Look at the new project structure:



- (d). You see two sub nodes in the Projects window. The first sub node, the "Modules" node, shows you the custom modules that are part of the application. Right now, as you can see, there are none. You can right-click on this sub node and then invoke wizards for creating new modules or for wrapping external JARs into the application. The "Important Files" node shows the build scripts and other supporting files used by the application.
- (e). Right-click the application and choose Run. The default splash screen is shown

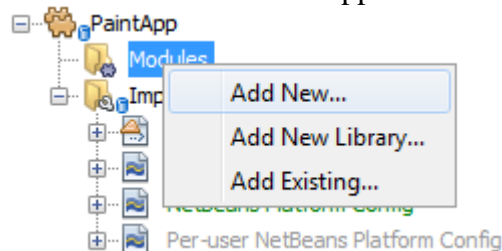
and then you see the starting point of your new application:



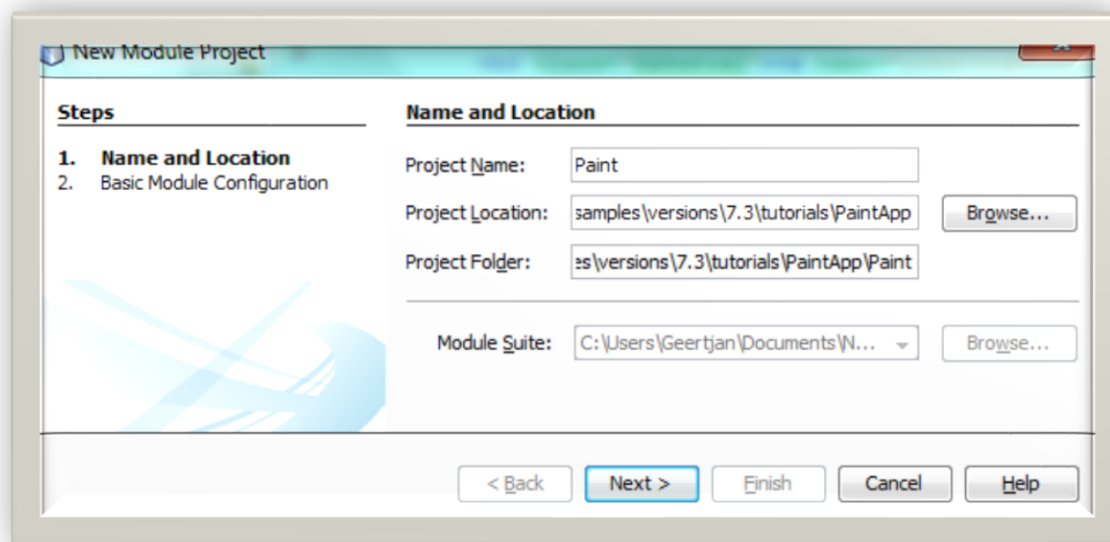
- (f). Look through the menu bar and the toolbar to see the features that your application already has, such as an Exit menu item, a Properties window, and an Output window.

7. **Creating the Functionality Module:** Now you need a module to contain the actual code you're going to write.

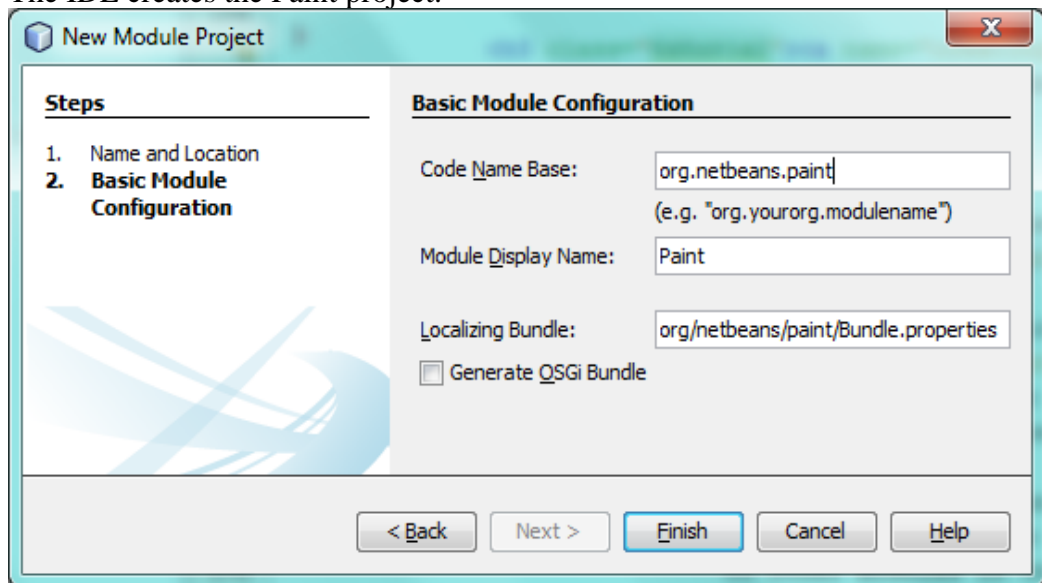
- (a). Right-click the "Modules" node in the Paint Application. Select Add New:



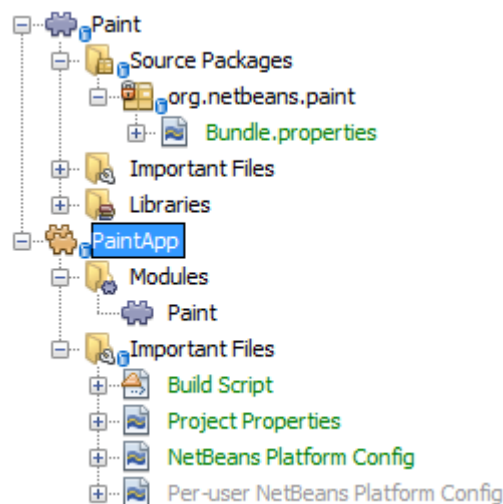
- (b). In the Name and Location panel, type **Paint** in Project Name. Notice that the module sources will be stored within a folder in the application's directory on disk. Click Next.



- (c). In the Basic Module Configuration panel, type org.netbeans.paint as the "Code Name Base". The code name base is a unique string identifying the module to other modules in the application. Leave everything unchanged. Click Finish. The IDE creates the Paint project.



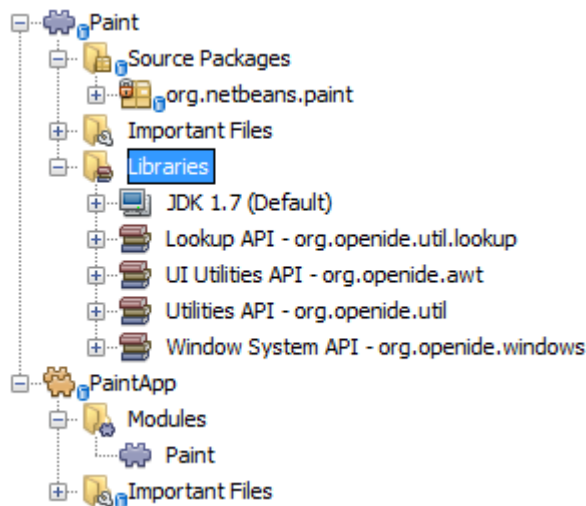
- (d). Take a look at the structure of your application. The project contains all of your sources and project metadata, such as the project's Ant build script. The project opens in the IDE. You can view its logical structure in the Projects window (Ctrl-1) and its file structure in the Files window (Ctrl-2). For example, the Projects window should look as follows:



- (e). You will need to subclass several classes that belong to the NetBeans APIs. All NetBeans APIs are implemented by modules, so this task really just means adding some modules to the list of modules that our module needs in order to run. In the Projects window, right-click the Paint project node and choose Properties. The Project Properties dialog box opens. Under Categories, click Libraries. For each of the API's listed in the table below, click "Add Dependency..." and then, in the Filter text box, start typing the name of the class that you want to subclass.

| Class        | API               | Purpose  |
|--------------|-------------------|--|
| Lookup       | Lookup API        | Enables loosely coupled communication between modules.   |
| ActionID     | UI Utilities API  | Provides annotations for registering Actions in the NetBeans Platform virtual filesystem, as well as the ColorComboBox class.            |
| Messages     | Utilities API     | Provides a variety of general utility classes, including support for internationalization via the Bundle class and @Messages annotation. |
| TopComponent | Window System API | Gives you access to the NetBeans window system.  |

- (f). The first column in the table above lists all the classes that you will subclass in this tutorial. In each case, start typing the class name in the Filter and watch the Module list narrow. Use the table's second column to pick the appropriate API (or, in the case of ColorChooser, the library) from the narrowed Module list and then click OK to confirm the choice. Click OK to exit the Project Properties dialog box.
- (g). In the Projects window, expand the Paint module's project node and then expand the Libraries node. Notice that all the libraries you have selected are displayed:



- (h). Expand the Paint module's Important Files node and double-click the Project Metadata node. Notice that the API's you selected have been declared as

module dependencies in the file. When the module is compiled, the declared dependencies are added to the module's manifest file.

8. **Creating the Canvas:** The next step is to create the actual component on which the user can paint. Here, you use a pure Swing component—so, let's skip the details of its implementation and just provide the final version. The color chooser bean, which you created the library wrapper module for, is used in the source code for this panel—when you run the finished application, you will see it in the toolbar of the panel for editing images.
  - (a). In the Projects window, expand the Paint node, then expand the Source Packages node, and then right-click the org.netbeans.paint node. Choose New > Java Class.
  - (b). Enter PaintCanvas as the Class Name. Ensure that org.netbeans.paint is listed as the Package. Click Finish. PaintCanvas.java opens in the Source editor.
  - (c). Replace the default content of the file with the content found below. If you named your package something other than org.netbeans.paint, correct the package name in the Source editor. Content is available here: <https://platform.netbeans.org/images/tutorials/paintapp/70/PaintCanvas.java>

```
package org.netbeans.paint;

import java.awt.Color;
import java.awt.Dimension;
import java.awt.Graphics;
import java.awt.Graphics2D;
import java.awt.Point;
import java.awt.RenderingHints;
import java.awt.event.MouseAdapter;
import java.awt.event.MouseEvent;
import java.awt.event.MouseMotionListener;
import java.awt.geom.AffineTransform;
import java.awt.image.BufferedImage;
import javax.swing.JComponent;

/**
 *
 * @author Tim Boudreau
 */
public class PaintCanvas extends JComponent {
    private int brushDiameter = 10;
```



```

private final MouseL mouseListener = new MouseL();
private BufferedImage backingImage = null;
private final BrushSizeView brushView = new BrushSizeView();
private Color color = Color.BLUE;

public PaintCanvas() {
    addMouseListener(mouseListener);
    addMouseMotionListener(mouseListener);
    setBackground(Color.WHITE);
    setFocusable(true);
}

public void setBrush(int diam) {
    this.brushDiameter = diam;
}

public void setBrushDiameter(int val) {
    this.brushDiameter = val;
    brushView.repaint();
}

public int getBrushDiameter() {
    return brushDiameter;
}

public void setColor(Color c) {
    this.color = c;
    brushView.repaint();
}

public Color getColor() {
    return color;
}

public void clear() {
    backingImage = null;
    repaint();
}

@Override
public void paint(Graphics g) {
    Graphics2D g2d = (Graphics2D) g;
    g2d.drawRenderedImage(getImage(), AffineTransform.getTranslateInstance(0,
0));
}

JComponent getBrushSizeView() {
    return brushView;
}

```

```

    }

    public BufferedImage getImage() {
        int width = Math.min(getWidth(), 1600);
        int height = Math.min(getHeight(), 1200);
        if (backingImage == null || backingImage.getWidth() != width ||
            backingImage.getHeight() != height) {
            int newWidth = backingImage == null ? width : Math.max(width,
                backingImage.getWidth());
            int newHeight = backingImage == null ? height : Math.max(height,
                backingImage.getHeight());
            if (newHeight > height && newWidth > width && backingImage != null) {
                return backingImage;
            }
            BufferedImage old = backingImage;
            backingImage = new BufferedImage(newWidth, newHeight,
                BufferedImage.TYPE_INT_ARGB_PRE);
            Graphics2D g = backingImage.createGraphics();
            g.setColor(Color.WHITE);
            g.fillRect(0, 0, width, height);
            if (old != null) {
                g.drawImage(old,
                    AffineTransform.getTranslateInstance(0, 0));
            }
            g.dispose();
            setPreferredSize(new Dimension (newWidth, newHeight));
        }
        return backingImage;
    }

    private class BrushSizeView extends JComponent {
        @Override
        public void paint(Graphics g) {
            ((Graphics2D) g).setRenderingHint(RenderingHints.KEY_ANTIALIASING,
                RenderingHints.VALUE_ANTIALIAS_ON);
            Point p = new Point(getWidth() / 2, getHeight() / 2);
            int half = getBrushDiameter() / 2;
            int diam = getBrushDiameter();
            g.setColor(getColor());
            g.fillOval(p.x - half, p.y - half, diam, diam);
        }

        @Override
        public Dimension getPreferredSize() {
            return new Dimension (24, 24);
        }
    }
}

```

```

private final class MouseL extends MouseAdapter implements
MouseListener {
    @Override
    public void mouseClicked(MouseEvent e) {
        Point p = e.getPoint();
        int half = brushDiameter / 2;
        Graphics2D g = getImage().createGraphics();
        g.setRenderingHint(RenderingHints.KEY_ANTIALIASING,
            RenderingHints.VALUE_ANTIALIAS_ON);
        g.setPaint(getColor());
        g.fillOval(p.x - half, p.y - half, brushDiameter, brushDiameter);
        g.dispose();
        repaint(p.x - half, p.y - half, brushDiameter, brushDiameter);
    }

    @Override
    public void mouseDragged(MouseEvent e) {
        mouseClicked(e);
    }
}

```

9. **Embedding the Canvas in a Window:** Now you'll write the only class in this application that needs to touch the NetBeans APIs. It is a [TopComponent](#) class. A TopComponent class is just a JPanel class which the NetBeans windowing system knows how to talk to—so that it can be put inside a tabbed container inside the main window.
  - (a). In the Projects window, expand the Paint node, then expand the Source Packages node, and then right-click the org.netbeans.paint node. Choose New > Java Class. Enter PaintTopComponent as the Class Name. Ensure that org.netbeans.paint is listed as the Package. Click Finish. PaintTopComponent.java opens in the Source editor.
  - (b). Near the top of the file, change the class declaration to the following:

```

public class PaintTopComponent extends TopComponent implements
ActionListener, ChangeListener {

```

- (c). **Ctrl-Shift-I** to fix imports and click OK. The IDE makes the necessary import package declarations at the top of the file:
 

```

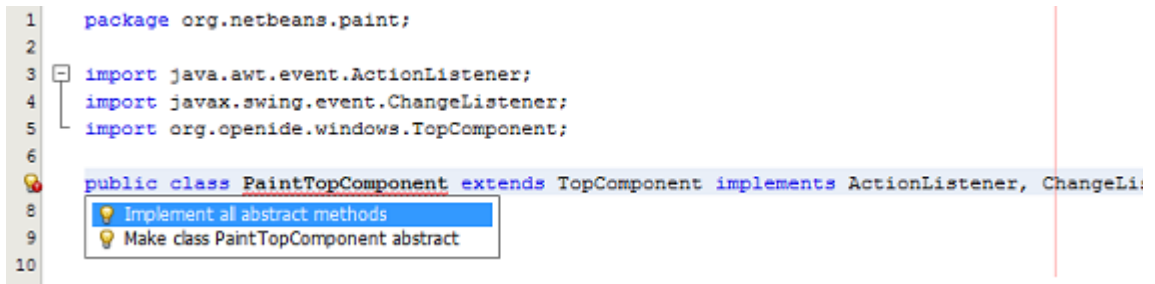
import java.awt.event.ActionListener;
import javax.swing.event.ChangeListener;
import org.openide.windows.TopComponent;

```
  - (d). Notice the red line under the class declaration that you just entered. Position the cursor in the line and notice that a light bulb appears in the left margin. Click the light bulb (or press Alt-Enter), as shown below:

```

1 package org.netbeans.paint;
2
3 import java.awt.event.ActionListener;
4 import javax.swing.event.ChangeListener;
5 import org.openide.windows.TopComponent;
6
7 public class PaintTopComponent extends TopComponent implements ActionListener, ChangeLi:
8
9
10

```



- (e). Select Implement all abstract methods. The IDE generates two method skeletons—actionPerformed() and stateChanged(). You will fill these out later in this tutorial.
- (f). Register the PaintTopComponent in the window system by adding annotations to the top of the class, as shown here, and then press Ctrl-Shift-I to let the IDE generate the appropriate import statements:

```

@TopComponent.Description(
    preferredID = "PaintTopComponent",
    iconBase = "/org/netbeans/paint/new_icon.png",
    persistenceType = TopComponent.PERSISTENCE_ALWAYS)
@TopComponent.Registration(
    mode = "editor",
    openAtStartup = true)
@ActionID(
    category = "Window",
    id = "org.netbeans.paint.PaintTopComponent")
@ActionReferences({
    @ActionReference(
        path = "Menu/Window",
        position = 0),
    @ActionReference(
        path = "Toolbars/File",
        position = 0)
})
@TopComponent.OpenActionRegistration(
    displayName = "#CTL_NewCanvasAction")
@Messages({
    "CTL_NewCanvasAction=New Canvas",
    "LBL_Clear=Clear",
    "LBL_Foreground=Foreground",
    "LBL_BrushSize=Brush Size",
    "# {0} - image",
    "UnsavedImageNameFormat=Image {0}"})
public class PaintTopComponent extends TopComponent implements

```

10. Add these two icons to "org/netbeans/paint": (save them in <path>\My Documents\NetBeansProjects\PaintApp\Paint\src\org\netbeans\Paint)



The 16x16 pixel icon will be used for the Small Toolbar Icons display, while the 24x24 pixel icon will be used for the Large Toolbar display, as well as in the tab of the window, as defined by @TopComponent.Description above.

11. Write the following code in PaintTopComponent class

```
private PaintCanvas canvas = new PaintCanvas(); //The component the user draws
on
    private final JComponent preview = canvas.getBrushSizeView(); //A component
in the toolbar that shows the paintbrush size
    private final JSlider brushSizeSlider = new JSlider(1, 24); //A slider to set the
brush size
    private final JToolBar toolbar = new JToolBar(); //The toolbar
    private final ColorComboBox color = new ColorComboBox(); //Our color
chooser component from the ColorChooser library
    private final JButton clear = new JButton(Bundle.LBL_Clear()); //A button to
clear the canvas
    private final JLabel label = new JLabel(Bundle.LBL_Foreground()); //A label for
the color chooser
    private final JLabel brushSizeLabel = new JLabel(Bundle.LBL_BrushSize());
//A label for the brush size slider
    private static int ct = 0; //A counter you use to provide names for new images

public PaintTopComponent() {
    initComponents();
    setDisplayName(Bundle.UnsavedImageNameFormat(ct++));
}

private void initComponents() {

    setLayout(new BorderLayout());

    //Configure our components, attach listeners:
    color.addActionListener(this);
    clear.addActionListener(this);
    brushSizeSlider.setValue(canvas.getBrushDiameter());
    brushSizeSlider.addChangeListener(this);
    color.setSelectedColor(canvas.getColor());
    color.setMaximumSize(new Dimension(16, 16));

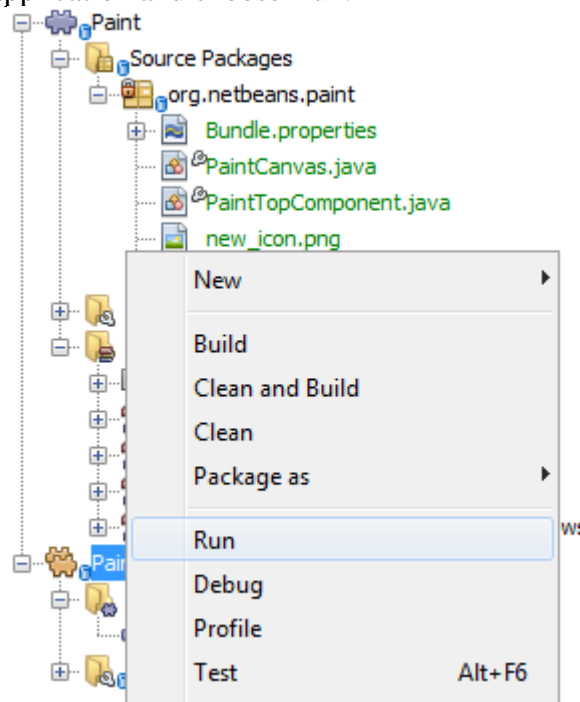
    //Install the toolbar and the painting component:
    add(toolbar, BorderLayout.NORTH);
    add(new JScrollPane(canvas), BorderLayout.CENTER);

    //Configure the toolbar:
    toolbar.setLayout(new FlowLayout(FlowLayout.LEFT, 7, 7));
```

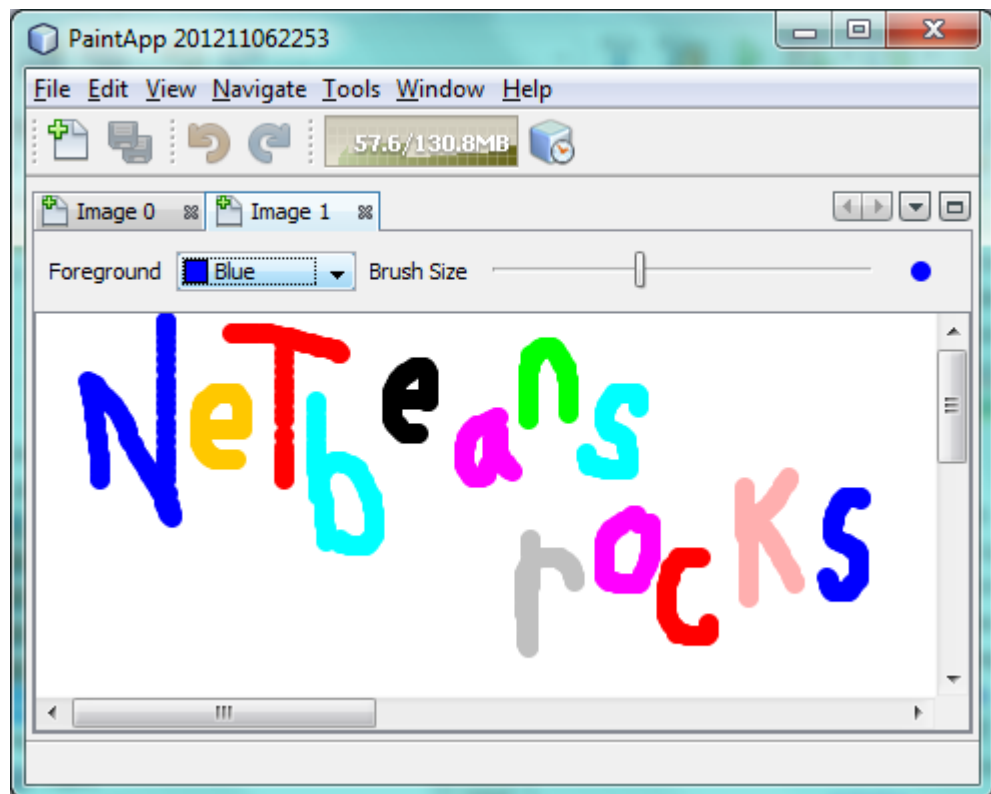
```
toolbar.setFloatable(false);
```

```
//Now populate our toolbar:  
toolbar.add(label);  
toolbar.add(color);  
toolbar.add(brushSizeLabel);  
toolbar.add(brushSizeSlider);  
toolbar.add(preview);  
toolbar.add(clear);
```

12. Write the following code in actionPerformed function  
if (e.getSource() instanceof JButton) {  
    canvas.clear();  
} else if (e.getSource() instanceof ColorComboBox) {  
    ColorComboBox cc = (ColorComboBox) e.getSource();  
    canvas.setColor(cc.getSelectedColor());  
}
13. Write the following line in stateChanged function  
    canvas.setBrushDiameter(brushSizeSlider.getValue());
14. **Running the Application:**
  - (a). Right-click the application and choose Run:



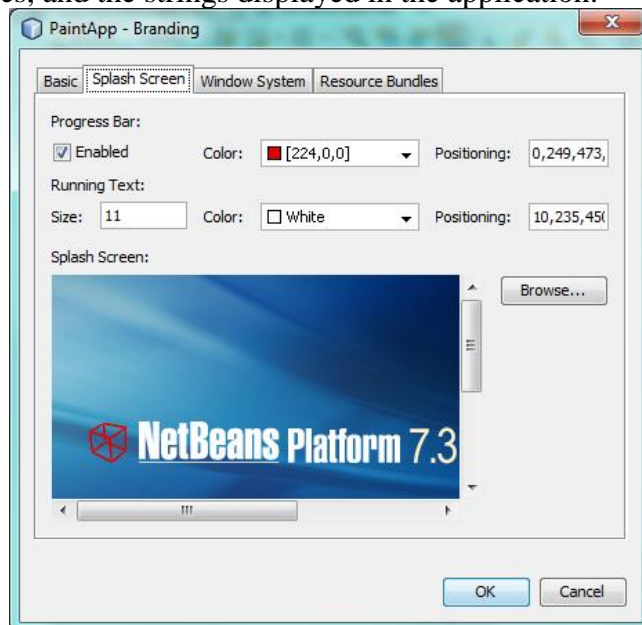
- (b). The application starts up, a splash screen is displayed, and then your application is shown. Paint something, as shown below:



(c). Use the application and try to identify areas where you'd like to provide more functionality.

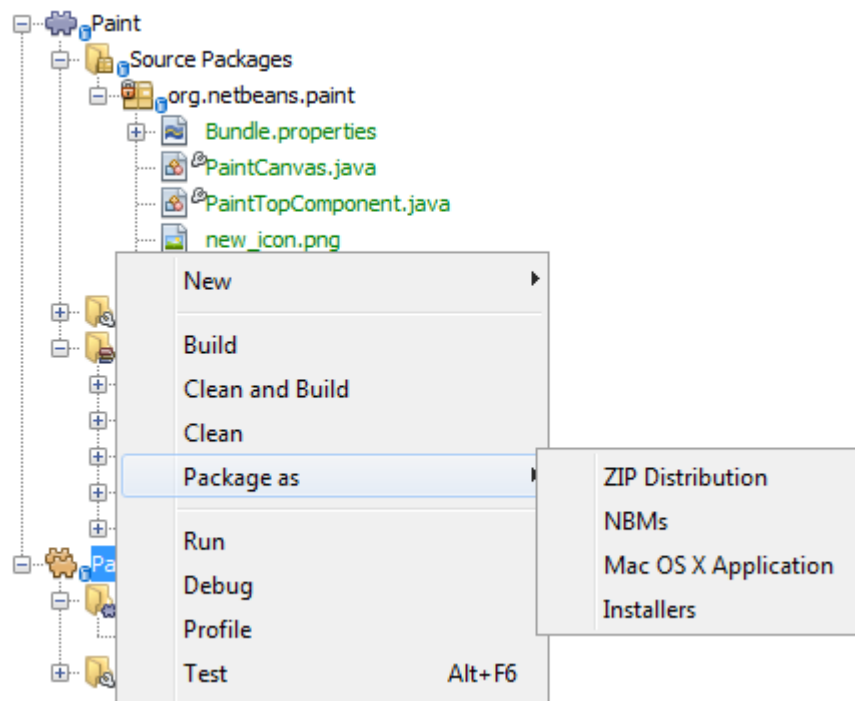
15. **Branding the Application:**

- (a). Right-click the application and choose Branding.
- (b). The Branding Window is shown, use it to change icons, the splash screen, the window features, and the strings displayed in the application:



16. **Packaging the Application**

- (a). Right-click the application and choose Package as:



- (b). Choose the distribution mechanism relevant to your business needs and your user requirements.
  - (c). Switch to the Files window (Ctrl-2) to see the result.
17. **Include following functionalities in the paint project: -** (10)
- (a). Eraser
  - (b). Line, rectangle and circle drawing mechanism
  - (c). Fill tool



**Web Resources:**

<https://platform.netbeans.org/tutorials/nbm-paintapp.html>

<https://platform.netbeans.org/tutorials/nbm-crud.html>

<https://java.net/projects/nb-api-samples/sources/api-samples/show/versions/7.3/tutorials/PaintApp>

<https://platform.netbeans.org/tutorials/nbm-google.html>

**Summary:** This lab allows the students to practice making a paint application using the built-in Netbeans modules.

## PROJECT –OBJECT ORIENTED PROGRAMMING - PHASE – I

**MAXIMUM MARKS:** 10

**DEADLINE:** After OHT - I

### **Instructions**

- This is a project assignment and is based on some of the lab experiments carried out by you.
- Plagiarism is strictly forbidden.
- You have to create the project for the given language specification.
- You have to show the working project for evaluation before deadline.
- The evaluation is based on project demonstration, working of the project for different test patterns and associated viva.
- Project will be graded as a syndicate effort.

1. **Objectives:**

- (a). Syndicate formation
- (b). Project proposal

2. **Programming Language:** Java

3. **Software Required:** NetBeans/Kawa/Eclipse/JCreator

4. **TASK**

You are required to create a project proposal identifying what kind of project you are going to develop. The project proposal must have the following

- (a). Abstract statement
- (b). Some functional requirements i.e., what the project will do
- (c). A snapshot of how will the GUI look like
- (d). Hard copy of proposal

## PROJECT –OBJECT ORIENTED PROGRAMMING - PHASE – II

### OOP Term Project Details

|                                    |                                |
|------------------------------------|--------------------------------|
| <b><u>MAXIMUM MARKS:</u></b>       | 20                             |
| <b><u>DEADLINE:</u></b>            | Before final lab               |
| <b><u>Programming Language</u></b> | Java                           |
| <b><u>Software Required:</u></b>   | NetBeans/Kawa/Eclipse/JCreator |

**Goal:** In this course our main goal is to learn how to build and evolve small to large-scale programs using object-oriented programming, and work in teams learning from each other.

**Topics:** In exploring object-oriented programming, we investigate three questions:

1. Design
2. Primitives
3. Implementation

In ***Design***, how do we think about a program in terms of objects? To answer this question, we explore CRC cards, UML, and design patterns.

In ***Primitives***, how do we express object orientation in terms of encapsulation, abstraction, inheritance and polymorphism? To answer this question, we explore classes, interfaces, method dispatch, method overriding, generics, operator overloading, and reflection, etc.

In ***Implementation***, how do we realize object-oriented primitives? To answer this question, we explore dynamic method dispatch, garbage collection and automatic memory management in detail.

Design and primitives matter because they represent the essence of object-oriented programming. Implementation matters because it enables us to debug object-oriented programs and tune their performance.

#### General Rules:

1. A group can be formed from 3 or 4 students per project.
2. Groups are not allowed to repeat ideas.
3. ***Groups can select projects from the given list or they are welcome to bring with their own ideas.***

**Submission Method:**

All project files are zipped and submitted named as “proj\_LeaderStudentID.zip”, where “LeaderStudentID” is replaced by a leader team member chosen by all team members. The zip file should contain:

1. All source code used to develop the project, either as a rar file, or a folder of the project packages and files.
2. A Design document contains a class diagram describing the project classes, and associations, and one paragraph describing the methods, one paragraph as a user manual, and one paragraph describing team member’s individual contributions.

**Grading Criteria:**

The project is worth 20% of your final mark. These marks are graded individually by asking each student in the presentation for their contribution and testing their understanding.

These are broken into:

1. 10 marks for correctness (no compilation or run-time errors).
2. 8 marks for the application of course concepts, where feasible, (such as: modularity, inheritance, polymorphism, method overriding and overloading, abstract classes & interfaces)
3. 2 marks for GUI interfaces, presentation, documentation, and teamwork.
4. 10 bonus marks for all other concepts you self-study outside the learning objectives of the course.

## List of Sample Projects

| Serial # | Name of Project                   |
|----------|-----------------------------------|
| 1        | Course Registration Application   |
| 2        | Chess Game                        |
| 3        | Text Editor                       |
| 4        | Online Quiz Application           |
| 5        | Document Management Application   |
| 6        | Bank Management Application       |
| 7        | Library Management Application    |
| 8        | Cafe Management Application       |
| 9        | Cyber Cafe Management Application |
| 10       | Project Management Application    |
| 11       | Graphics Editor Application       |
| 12       | Staff Management Application      |
| 13       | Brick Game                        |
| 14       | Soft piano                        |
| 15       | Paranoid Game                     |

## Sample Project Details

### 1. Bank Management System:

Develop an application to help a bank manager manage customer accounts. The bank offer several bank accounts types. Each customer can have one or more accounts. The customer can go to the operations permitted by the account type, such as deposit, withdraw, or balance enquire. The bank manages the account by debiting the fees, or crediting the profits. Both the bank employees and the customers can print reports about the current account details.

#### Design:

*Basic Classes:* Account, CheckingAccount, SavingAccount, Loan, Customer:

- A. The Account is a general account class that contains balance as instance variable, deposit, withdraw, and balanceEnquiry as instance methods.
- B. Checking Account is a subclass from the Account class that allows overdraft while withdrawing (making the balance go below zero up to the specified credit limit), by debiting the account balance with an overdraft fee. It has a creditlimit as an instance variable.
- C. The Saving Account is a subclass from the Account class that has an interest rate as an instance variable. The system credit the balance with monthly interest based on the account balance and the interest rate.
- D. The loan account is a subclass from the Account class that has principal amount, interest rate, loan duration in months as instance variables. The loan balance is debited by monthly interest each month based on the interest rate and the loan balance.
- E. Each customer can have any number of accounts of any type.
- F. The banking system provide the customer with an interface to access all banking operations described above and review reports about transactions and current balance.
- G. A banking administrator can print a report about all customers and their current balances.

Provides an interface for the user to:

- 1. Adding/editing/deleting GUI to each class,
- 2. GUI for Customers to open a new account
- 3. GUI for Customers to view transactions and balance for all their accounts.
- 4. GUI for a administrator to view all customer balances.

Sample data include:

**customers.txt**

| Customer ID | Name    | Address | Phone | Email |
|-------------|---------|---------|-------|-------|
| 1           | Mohamed | ...     | ...   | ...   |
| 2           | Ahmed   | ...     | ...   | ...   |
| 3           | Mostafa | ...     | ...   | ...   |

**accounts.txt**

| Account ID | Customer ID | Type                | Balance | CreditLimit or<br>Interest Rate | Principal<br>Amount | Loan<br>Duration |
|------------|-------------|---------------------|---------|---------------------------------|---------------------|------------------|
| 1          | 1           | Saving              | ...     |                                 |                     |                  |
| 2          | 1           | Loan                | ...     |                                 |                     |                  |
| 3          | 2           | Checking<br>Account | ...     |                                 |                     |                  |

**accountTransactions.txt**

| Account ID | Date Time | Transaction Type | Amount | Transaction Type |
|------------|-----------|------------------|--------|------------------|
| 1          | 9/1/2013  | Withdraw         | ...    | Withdraw         |
| 1          | ...       | Deposit          | ...    | Rowing machine   |
| 2          | ...       | Interest         | ...    | Ab Roller        |
| 3          | ...       | Fees             | ...    |                  |

**“I hear and I forget,  
I see and I remember,  
I do and I understand”**

**Confucius**



**“A scientist in his laboratory is not a mere technician: he is also a child confronting natural phenomena that impress him as though they were fairy tales.”**

**Marie Curie**

