LAB MANUAL

FOR

B.TECH PROGRAM

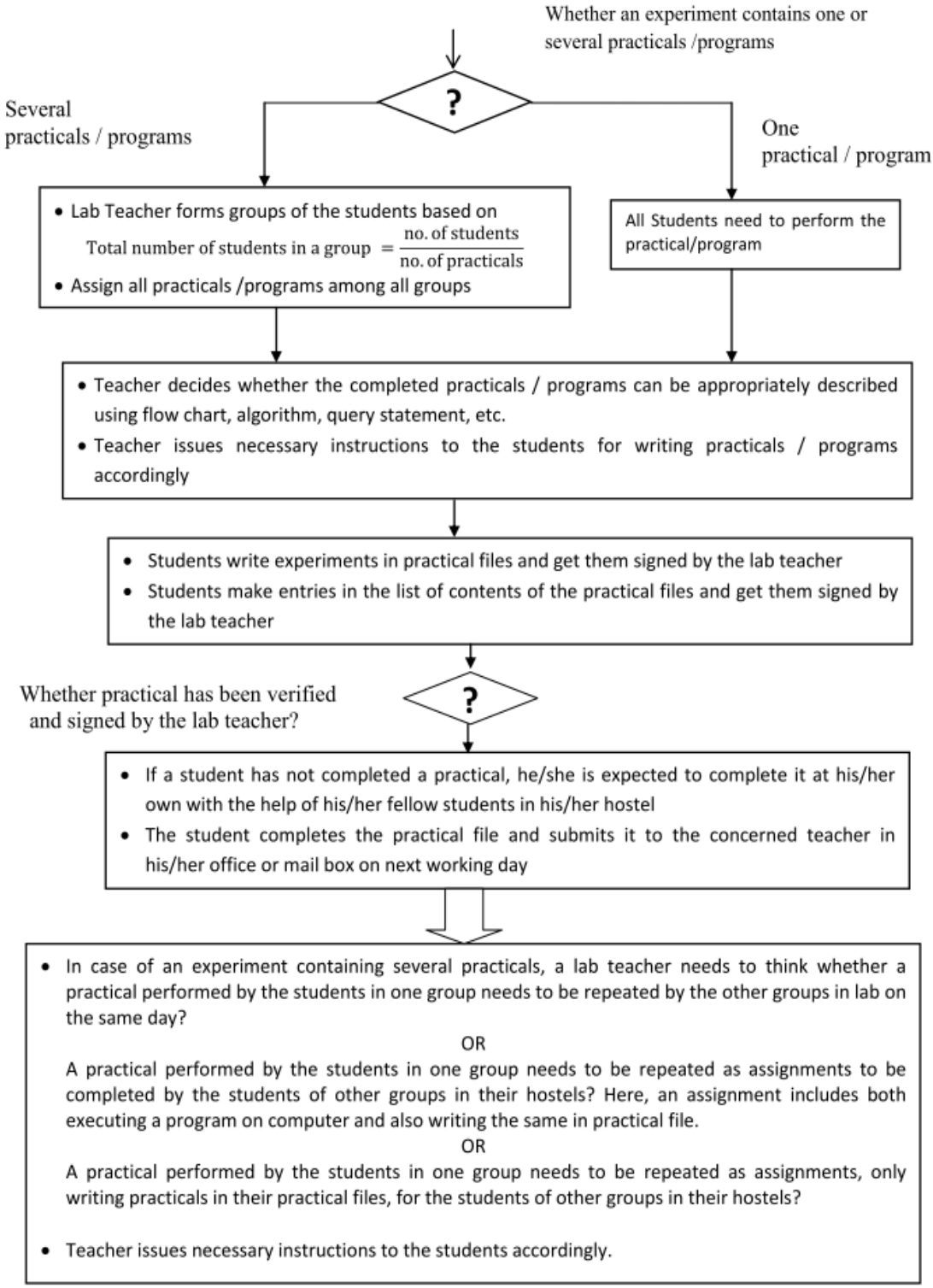
(2nd year)

JAVA PROGRAMMING

ITPC-23

NATIONAL INSTITUTE OF TECHNOLOGY KURUKSHETRA

Lab Instructions



Course Code	**:**	**ITPC23**
Course Title	**:**	**Java Programming**
Number of Credits	**:**	**5**
Prerequisites	**:**	**CSIR11**
Course Type	**:**	**PC**

Course Learning Objectives:

1. Building robust applications using Java's object-oriented features.
2. Understanding the usage of java class libraries.
3. Building multithreaded, platform-independent and GUI based java applications for business problems.

Course Content:

1. The overview of Java's architecture and the architecture of the Java Virtual Machine (JVM).
   Classes: Declaring Members (Fields and Methods), Instance Members, Static Members.
   Objects: Class Instantiation, Reference Values, and References, Object Aliases. Basic Language Elements, Primitive Data Types, Variable Declarations, Initial Values for Variables, Class Declarations, Method Declarations, this reference, Method Overloading, Constructors, The Default Constructor and Constructors overloading. Arrays, Anonymous Arrays, Multidimensional Arrays, Variable Arity Methods, The main() Method, Program Arguments.

2. **Packages:** Defining Packages, Using Packages, Compiling Code into Packages, Running Code from Packages. Scope Rules, Accessibility Modifiers, Overview of other Modifiers for Members.
   Operators and Expressions**,** Overview of Control Flow Statements.
   Exception Handling: The try Block, The catch Block, The finally Block, The throw Statement, The throws Clause, Checked and Unchecked Exceptions, Defining New Exceptions.

3. **Object-Oriented Programming:** Single Implementation Inheritance, Overriding Methods, Hiding Members, The Object Reference super, Chaining Constructors Using this() and super()
   Interfaces: Defining Interfaces, Abstract Method Declarations, Implementing Interfaces, Extending Interfaces, Interface References, Constants in Interfaces, Polymorphism and Dynamic Method Lookup.
   Fundamental Classes: Overview of the java.lang Package, The Object Class, The Wrapper Classes, The String Class, The StringBuilder and the StringBuffer Classes.

4. **Multithreading:** Overview of Threads, the Main Thread, Thread Creation, Synchronization, Thread Transitions. Basics of Event Handling, Graphics Programming using AWT and Swing, An overview of the new Features of Java 7 & 8.

Reference Books:

1. Bruce Eckel, *Thinking In Java,* Pearson Education, 4th Ed., 2006.
2. Dietel & Deitel, *Java How to Program*, Pearson Education, 10th Ed., 2015.
3. Kathy Sierra & Bert Bates, *Head First Java*, O'REILLY, 2nd Ed., 2005.
4. Cay s. Horstmann & Gary Cornell, *Core Java. Volume I, Fundamentals,* Sun Microsystems Press, 8th Ed., 2008.

Course outcomes:

1. Write Java programs that solve simple business problems.
2. Create java applications that are robust and multithreaded.
3. Write simple GUI interfaces for a program to interact with users, and to understand the event-based GUI handling principles.

JAVA PROGRAMMING LAB (ITPC-23)

EXPERIMENT 1:

1. Create a class containing a **float** and use it to demonstrate aliasing.
2. Write a program that uses two nested **for** loops and the modulus operator (**%**) to detect and print prime numbers (integral numbers that are not evenly divisible by any other numbers except for themselves and 1).
3. Create a **switch** statement that prints a message for each **case**, and put the **switch** inside a **for** loop that tries each **case**. Put a **break** after each **case** and test it, then remove the **break**s and see what happens.
4. Create a class containing an uninitialized **String** reference. Demonstrate that this reference is initialized by Java to **null**.

EXPERIMENT 2:

1. Write a method that takes two **String** arguments and uses all the **boolean** comparisons to compare the two **String**s and print the results. For the == and **!**=, also perform the **equals( )** test. In **main( )**, call your method with some different **String** objects.
2. Create a class called **Dog** containing two **String**s: **name** and **says**. In **main( )**, create two dog objects with names "spot" (who says, "Ruff!") and "scruffy" (who says, "Wurf!"). Then display their names and what they say. Also create a new **Dog** reference and assign it to spot's object. Test for comparison using == and **equals( )** for all references.

EXPERIMENT 3:

1. Create a class called **Dog** with an overloaded **bark( )** method. This method should be overloaded based on various primitive data types, and print different types of barking, howling, etc., depending on which overloaded version is called. Write a **main( )** that calls all the different versions.
2. Create a class without a constructor, and then create an object of that class in **main( )** to verify that the default constructor is automatically synthesized.
3. Create a class with two (overloaded) constructors. Using **this**, call the second constructor inside the first one.

EXPERIMENT 4:

1. Create a class with a **static String** field that is initialized at the point of definition, and another one that is initialized by the **static** block. Add a **static** method that prints both fields and demonstrates that they are both initialized before they are used.
2. Create a **main( )** that uses varargs instead of the ordinary **main( )** syntax. Print all the elements in the resulting **args** array. Test it with various numbers of command-line arguments.

EXPERIMENT 5:

1. Create a class with **public**, **private**, **protected**, and package-access fields and method members. Create an object of this class and see what kind of compiler messages you get when you try to access all the class members. Be aware that classes in the same directory are part of the "default" package.

EXPERIMENT 6:

1. Create two classes, **A** and **B**, with default constructors (empty argument lists) that announce themselves. Inherit a new class called **C** from **A**, and create a member of class **B** inside **C**. Do not create a constructor for **C**. Create an object of class **C** and observe the results.
2. Create a base class with only a non-default constructor, and a derived class with both a default (no-arg) and non-default constructor. In the derived-class constructors, call the base-class constructor.
3. Create a class with a method that is overloaded three times. Inherit a new class, add a new overloading of the method, and show that all four methods are available in the derived class.
4. Create a class with a **final** method. Inherit from that class and attempt to overwrite that method. **(ii)** Create a **final** class and attempt to inherit from it.

EXPERIMENT 7:

1. Create a base class with an **abstract print( )** method that is overridden in a derived class. The overridden version of the method prints the value of an **int** variable defined in the derived class. At the point of definition of this variable, give it a nonzero value. In the base-class constructor, call this method. In **main( )**, create an object of the derived type, and then call its **print( )** method. Explain the results.
2. Write a program to demonstrate that all the methods in an interface are automatically **public.**

3. Create three interfaces, each with two methods. Inherit a new interface that combines the three, adding a new method. Create a class by implementing the new interface and also inheriting from a concrete class. Now write four methods, each of which takes one of the four interfaces as an argument. In **main( )**, create an object of your class and pass it to each of the methods.

EXPERIMENT 8:

1. Create a class with a **private** field and a **private** method. Create an inner class with a method that modifies the outer-class field and calls the outer-class method. In a second outer-class method, create an object of the inner class and call its method, then show the effect on the outer-class object.
2. Determine whether an outer class has access to the **private** elements of its inner class.

EXPERIMENT 9:

1. Create a class with a **main( )** that throws an object of class **Exception** inside a **try** block. Give the constructor for **Exception** a **String** argument. Catch the exception inside a **catch** clause and print the **String** argument. Add a **finally** clause and print a message to prove you were there.
2. Create your own exception class using the **extends** keyword. Write a constructor for this class that takes a **String** argument and stores it inside the object with a **String** reference. Write a method that displays the stored **String**. Create a **try-catch** clause to exercise your new exception.

REFERENCES:

1. Thinking in Java, 4th Ed, Bruce Eckel
2. Sun Microsystems Core Java, Vol. I & II
3. Java Tutorials on : www.oracle.com