**OOC IAT-1 Solution**

**1(a). Define Method Overloading. Write a JAVA program with three overloaded function area() to find area of rectangle, area of rectangular box, area of circle**

.

**Method Overloading** is a feature that allows a class to have more than one **method** having the same name, if their argument lists are different.

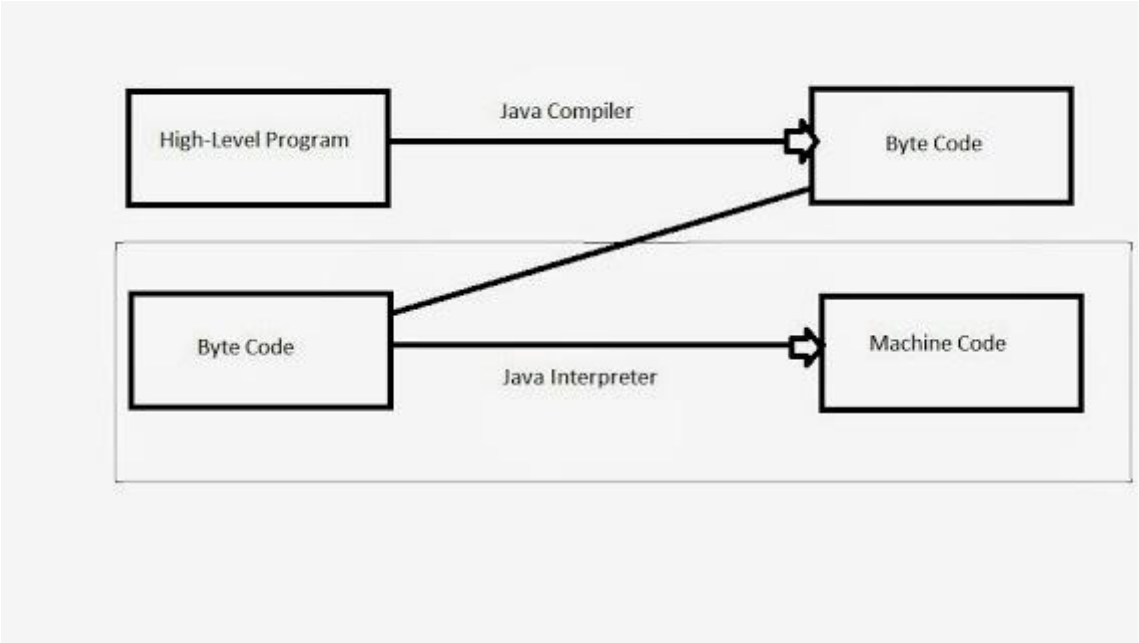
```
1.  class OverloadDemo
2.  {
3.     void area(float x)
4.     {
5.        System.out.println("the area of the square is "+Math.pow(x, 2)+" sq
    units");
6.     }
7.     void area(float x, float y)
8.     {
9.        System.out.println("the area of the rectangle is "+x*y+" sq units");
10.    }
11.    void area(double x)
12.    {
13.       double z = 3.14 * x * x;
14.       System.out.println("the area of the circle is "+z+" sq units");
15.    }
16. }
17. class Overload
18. {
19.     public static void main(String args[])
20.        {
21.          OverloadDemo ob = new OverloadDemo();
22.          ob.area(5);
23.          ob.area(11,12);
24.          ob.area(2.5);
25.       }
26. }
```

**2(a). Why java is known as a Platform-neutral (independent) Language? Elaborate.**

Java is known as platform independent because it uses the concept of generating the byte code of the high level program ,and then run that byte code(intermediate code) on JVM(Java Virtual Machine),now what is JVM, actual JVM is a virtual computer system that run on your original computer system .

JVM converts the byte code to machine code according to the your original computer's

machine architecture. Here is flow how the Java handle your high level program.




So java compilation is done only once ,after that the byte code can be interpreted on any machine that have JVM.JVM is of different type according to computer system architecture,means for x86 JVM will be different for ARM JVM will be different etc.These are developed by the java vender that is Oracle.This JVM come as a part of JDK(Java Development Kit). So this compiler and interpreter has make it platform-neutral language.

**2(b). Write short note on**
**i) this keyword**

Sometimes a method will need to refer to the object that invoked it. To allow this, Java defines the **this** keyword. **this** can be used inside any method to refer to the *current* object. That is, **this** is always a reference to the object on which the method was invoked. You can use **this** anywhere a reference to an object of the current class' type is permitted.
To better understand what **this** refers to, consider the following version of **Box( )**:

```
// A redundant use of this.
Box(double w, double h, double d) {
this.width = w;
this.height = h;
this.depth = d;
}
```

**ii) for each.**
**The For-Each Version of the for Loop**
The for-each style of **for** is also referred to as the *enhanced* **for** loop.

The general form of the for-each version of the **for** is shown here:
for(*type itr-var : collection*) *statement-block*
Here, *type* specifies the type and *itr-var* specifies the name of an *iteration variable* that will receive the elements from a collection, one at a time, from beginning to end. The collection being cycled through is specified by *collection*

```
// Use a for-each style for loop.
class ForEach {
public static void main(String args[]) {
int nums[] = { 1, 2, 3, 4, 5, 6, 7, 8, 9, 10 };
int sum = 0;
// use for-each style for to display and sum the values
for(int x : nums) {
System.out.println("Value is: " + x);
sum += x;
}
System.out.println("Summation: " + sum);
}
}
```

**3(a). What is super? Explain the Significance of super (both constructor and method) with example**

super is a keyword.

- It is used inside a sub-class method definition to call a method defined in the super class. Private methods of the super-class cannot be called. Only public and protected methods can be called by the super keyword.
- It is also used by class constructors to invoke constructors of its parent class.

Syntax:

```
super.<method-name>([zero or more arguments]);
```

or:

```
super([zero or more arguments]);
```

**Code listing 1: SuperClass.java**

```
1 public class SuperClass {
2   public void printHello() {
3     System.out.println("Hello from SuperClass");
4     return;
5   }
6 }
```

**Code listing 2: SubClass.java**

```
1 public class SubClass extends SuperClass {
2   public void printHello() {
3     super.printHello();
4     System.out.println("Hello from SubClass");
5     return;
6   }
7   public static main(String[] args) {
8     SubClass obj = new SubClass();
9     obj.printHello();
10  }
11 }
```

**4(a). List and explain the java BUZZWORDS?**

The Java Buzzwords
Simple
• Secure
• Portable
• Object-oriented
• Robust
• Multithreaded
• Architecture-neutral
• Interpreted
• High performance
• Distributed
• Dynamic

Two of these buzzwords have already been discussed: secure and portable. Let's examine what each of the others implies.

**Simple:** Java was designed to be easy for the professional programmer to learn effectively. According to Sun, Java language is simple because:

syntax is based on C++ (so easier for programmers to learn it after C++).

removed many confusing and/or rarely-used features e.g., explicit pointers, operator ove

etc.

No need to remove unreferenced objects because there is Automatic Garbage Collection in java.

**Object-Oriented:** Object-oriented means we organize our software as a combination of different types of objects that incorporates both data and behavior.

Object-oriented programming(OOPs) is a methodology that simplify software development and maintenance by providing some rules.

**Robust:** Robust simply means strong. Java uses strong memory management. There are lack of pointers that avoids security problem. There is automatic garbage collection in java. There is exception handling and type checking mechanism in java. All these points makes java robust.

**Multithreaded:** A thread is like a separate program, executing concurrently. We can write Java programs that deal with many tasks at once by defining multiple threads. The main advantage of multi-threading is that it doesn't occupy memory for each thread. It shares a common memory area. Threads are important for multi-media, Web applications etc.

**Architecture-Neutral:** There are no implementation dependent features e.g. size of primitive types is fixed.In C programming, int data type occupies 2 bytes of memory for 32-bit architecture and 4 bytes of memory for 64-bit architecture. But in java, it occupies 4 bytes of memory for both 32 and 64 bit architectures.

**Portable:** We may carry the java bytecode to any platform.

**High Performance:** Java is faster than traditional interpretation since byte code is "close" to native code still somewhat slower than a compiled language (e.g., C++)

**Distributed:** We can create distributed applications in java. RMI and EJB are used for creating distributed applications. We may access files by calling the methods from any machine on the internet.

**Dynamic:** Java programs carry with them substantial amounts of run-time type information that is used to verify and resolve accesses to objects at run time. This makes it possible to dynamically link code in a safe and expedient manner. This is crucial to the robustness of the Java environment, in which small fragments of bytecode may be dynamically updated on a running system.

**4(b). Discuss three OOP principles.**

The Three OOP Principles

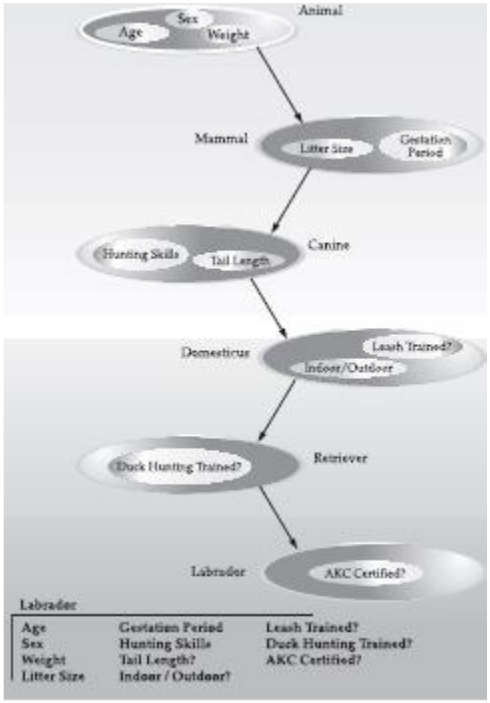
All object-oriented programming languages provide mechanisms that help you implement the object-oriented model. They are encapsulation, inheritance, and

polymorphism. Let's take a look at these concepts now.

**Encapsulation:**
*Encapsulation* is the mechanism that binds together code and the data it manipulates, and keeps both safe from outside interference and misuse. One way to think about encapsulation is as a protective wrapper that prevents the code and data from being arbitrarily accessed by other code defined outside the wrapper. Access to the code and data inside the wrapper is tightly controlled through a well-defined interface. To relate this to the real world, consider the automatic transmission on an automobile. It encapsulates hundreds of bits of information about your engine, such as how much you are accelerating, the pitch of the surface you are on, and the position of the shift lever. You, as the user, have only one method of affecting

**Inheritance:**
*Inheritance* is the process by which one object acquires the properties of another object. This is important because it supports the concept of hierarchical classification. As mentioned earlier, most knowledge is made manageable by hierarchical (that is, top-down) classifications.
For example, a Golden Retriever is part of the classification *dog*, which in turn is part of the *mammal* class, which is under the larger class *animal*. Without the use of hierarchies, each object would need to define all of its characteristics explicitly. However, by use of inheritance, an object need only define those qualities that make it unique within its class. It can inherit its general attributes from its parent. Thus, it is the inheritance mechanism that makes it possible for one object to be a specific instance of a more general case.

**Polymorphism:**
*Polymorphism* (from Greek, meaning "many forms") is a feature that allows one interface to be used for a general class of actions. The specific action is determined by the exact nature




of the situation. Consider a stack (which is a last-in, first-out list). You might have a program that requires three types of stacks. One stack is used for integer values, one for floating-point values, and one for characters. The algorithm that implements each stack is the same, even though the data being stored differs. In a non–object-oriented language, you would be required to create three different sets of stack routines, with each set using different names.
However, because of polymorphism, in Java you can specify a general set of stack routines that all share the same names.

**5(a). Define Constructor. When it is invoked. Explain with an example.**
Java constructor: A constructor in Java is a method which is used used to initialize objects. Constructor method of a class has the same name as that of the class, they are called or invoked when an object of a class is created and can't be called explicitly. Attributes of an object may or may not be available while creating objects, if no attribute is available then default constructor is called, some of the attributes may be known initially. It is optional to write constructor method(s) in a class but due to their utility they are used. It will be invoked while creating the objects.

**Java constructor example**

---

```
class Programming {

  //constructor method

  Programming() {
```

```
   System.out.println("Constructor method called.");

  }

  public static void main(String[] args) {

   Programming object = new Programming(); // Creating an object and invoking
constructore

  }

}
```

---

**5(b). EXPLAIN the following Operators: a) >>> b) short circuit logical operators.**

**The Unsigned Right Shift**

As you have just seen, the >> operator automatically fills the high-order bit with its previous contents each time a shift occurs. This preserves the sign of the value. However, sometimes this is undesirable. For example, if you are shifting something that does not represent a numeric value, you may not want sign extension to take place. This situation is common when you are working with pixel-based values and graphics. In these cases, you will generally want to shift a zero into the high-order bit no matter what its initial value was. This is known as an *unsigned shift*. To accomplish this, you will use Java's unsigned, shift-right operator, >>>, which always shifts zeros into the high-order bit.

The following code fragment demonstrates the >>>. Here, **a** is set to –1, which sets all 32 bits to 1 in binary. This value is then shifted right 24 bits, filling the top 24 bits with zeros, ignoring normal sign extension. This sets **a** to 255.

```
int a = -1;
a = a >>> 24;
```

Here is the same operation in binary form to further illustrate what is happening:

11111111 11111111 11111111 11111111 –1 in binary as an int
>>>24
00000000 00000000 00000000 11111111 255 in binary as an int

**Short-Circuit Logical Operators**

Java provides two interesting Boolean operators not found in many other computer languages.

These are secondary versions of the Boolean AND and OR operators, and are known as *short-circuit* logical operators. As you can see from the preceding table, the OR operator results in **true** when **A** is **true**, no matter what **B** is. Similarly, the AND operator results in **false** when **A** is **false**, no matter what **B** is. If you use the **||** and **&&** forms, rather than the **|** and **&** forms of these operators, Java will not bother to evaluate the right-hand operand when the outcome of the expression can be determined by the left operand alone. This is very useful when the right-hand operand depends on the value of the left

one in order to function properly. For example, the following code fragment shows how you can take advantage of short-circuit logical evaluation to be sure that a division operation will be valid before evaluating it:
if (denom != 0 && num / denom > 10)
For example, consider the following statement:
if(c==1 & e++ < 100) d = 100;
Here, using a single **&** ensures that the increment operation will be applied to **e** whether **c** is equal to 1 or not.

**6(a). Write a java program to represent planets in the solar system. Each planet has fields for the planets name, its distance from the sun in miles and the number of moons it has. Write a program to read the data for each planet and display.**

```
import java.util.Scanner;

class Planet{
	String name;
	int distanceInMiles;
	int moons;
	void getdata() {
		Scanner s = new Scanner(System.in);
		System.out.println("enter the planet name : ");
		name = s.next();
		System.out.println("enter the distanceFrom Sun : ");
		distanceInMiles = s.nextInt();
		System.out.println("number of moons : ");
		moons = s.nextInt();
	}

	void display() {
		System.out.println("name of the planet : "+name);
		System.out.println("the distanceFrom Sun : "+distanceInMiles);
		System.out.println("number of moons : "+moons);
	}
}
public class SolarSystem {
	public static void main(String[] args) {
		Planet planet[] =new Planet[8];
		for (int i = 0; i < 8; i++)
		 planet[i] = new Planet();

		for (int i = 0; i < 8; i++) {
			planet[i].getdata();
		}

			for (int i = 0; i < 8; i++) {
				planet[i].display();
			}

	}

}
```

**7(a). Define an Exception. Write a JAVA program to Demonstrate Exception Handling.**

In java, exception is an event that disrupts the normal flow of the program. It is an object which is thrown at runtime.

**Arithmetic exception**

```
// Java program to demonstrate ArithmeticException
class ArithmeticException_Demo
{
   public static void main(String args[])
   {
      try {
         int a = 30, b = 0;
         int c = a/b;  // cannot divide by zero
         System.out.println ("Result = " + c);
      }
      catch(ArithmeticException e) {
         System.out.println ("Can't divide a number by 0");
      }
   }
}
```

**7(b). List and explain the eight basic Data types used in Java. Give examples.**

Primitive Data Types

There are eight primitive datatypes supported by Java. Primitive datatypes are predefined by the language and named by a keyword. Let us now look into the eight primitive data types in detail.

byte

- Byte data type is an 8-bit signed two's complement integer
- Minimum value is -128 (-2^7)
- Maximum value is 127 (inclusive)(2^7 -1)
- Default value is 0
- Byte data type is used to save space in large arrays, mainly in place of integers, since a byte is four times smaller than an integer.
- Example: byte a = 100, byte b = -50

short

- Short data type is a 16-bit signed two's complement integer
- Minimum value is -32,768 (-2^15)
- Maximum value is 32,767 (inclusive) (2^15 -1)
- Short data type can also be used to save memory as byte data type. A short is 2 times smaller than an integer
- Default value is 0.

- Example: short s = 10000, short r = -20000

int

- Int data type is a 32-bit signed two's complement integer.
- Minimum value is - 2,147,483,648 (-2^31)
- Maximum value is 2,147,483,647(inclusive) (2^31 -1)
- Integer is generally used as the default data type for integral values unless there is a concern about memory.
- The default value is 0
- Example: int a = 100000, int b = -200000

long

- Long data type is a 64-bit signed two's complement integer
- Minimum value is -9,223,372,036,854,775,808(-2^63)
- Maximum value is 9,223,372,036,854,775,807 (inclusive)(2^63 -1)
- This type is used when a wider range than int is needed
- Default value is 0L
- Example: long a = 100000L, long b = -200000L

float

- Float data type is a single-precision 32-bit IEEE 754 floating point
- Float is mainly used to save memory in large arrays of floating point numbers
- Default value is 0.0f
- Float data type is never used for precise values such as currency
- Example: float f1 = 234.5f

double

- double data type is a double-precision 64-bit IEEE 754 floating point
- This data type is generally used as the default data type for decimal values, generally the default choice
- Double data type should never be used for precise values such as currency
- Default value is 0.0d
- Example: double d1 = 123.4

boolean

- boolean data type represents one bit of information
- There are only two possible values: true and false

- This data type is used for simple flags that track true/false conditions
- Default value is false
- Example: boolean one = true

char

- char data type is a single 16-bit Unicode character
- Minimum value is '\u0000' (or 0)
- Maximum value is '\uffff' (or 65,535 inclusive)
- Char data type is used to store any character
- Example: char letterA = 'A'