**INTERNITY FOUNDATION**

**TASK 3**

**Submitted By:**

**Amisha Singhal**

**Java Batch**

**Ques :Working with Methods and Encapsulation  
Create methods with arguments and return values; including overloaded methods**

**Ans-**

**Creating Methods:**

Example of a method declaration:

public double calculateAnswer(double wingSpan, int numberOfEngines,

double length, double grossTons) {

//do the calculation here

}

The only required elements of a method declaration are the method’s return type, name, a pair of parentheses, (), and a body between braces, {}.

Method declarations have six components, in order:

1. Modifiers—such as public, private, and others you will learn about later.
2. The return type—the data type of the value returned by the method, or void if the method does not return a value.
3. The method name—the rules for field names apply to method names as well, but the convention is a little different.
4. The parameter list in parenthesis—a comma-delimited list of input parameters, preceded by their data types, enclosed by parentheses, (). If there are no parameters, you must use empty parentheses.
5. An exception list—to be discussed later.
6. The method body, enclosed between braces—the method’s code, including the declaration of local variables, goes here.

#### Return statement

A return statement is used to exit from a method, with or without a value. For methods that define a return type, the return statement must be immediately followed by a return value. For methods that don’t return a value, the return statement can be used without a return value to exit a method.

The return statement need not be the *last statement* in a method, but it must be the *last statement to execute* in a method:

void setWeight(double val) {

if (val < 0)

return;

else

weight = val;

}

### Overloaded method

*Overloaded methods a*re methods with the same name but different method parameter lists. Java can distinguish between methods with different *method signatures*. This means that methods within a class can have the same name if they have different parameter lists.

Eg:

**class** Adder{

**static** **int** add(**int** a,**int** b){**return** a+b;}

**static** **int** add(**int** a,**int** b,**int** c){**return** a+b+c;}

}

**class** TestOverloading1{

**public** **static** **void** main(String[] args){

System.out.println(Adder.add(11,11));

System.out.println(Adder.add(11,11,11));

}}

**Ques- Apply the static keyword to methods and fields.**

**Ans-**

**Java static variable**

* It is a variable which **belongs to the class** and **not**to **object**(instance)
* Static variables are **initialized only once**, at the start of the execution . These variables will be initialized first, before the initialization of any instance variables
* A **single copy** to be shared by all instances of the class
* A static variable can be **accessed directly**by the **class name** and doesn’t need any object
* Syntax : <***class-name>.<variable-name>***

**Java Static Method**

* It is a method which **belongs to the class**and **not**to the **object**(instance)
* A static method **can access only static data**. It cannot access non-static data (instance variables)
* A static method **can call** **only** other **static methods**and cannot call a non-static method from it.
* A static method can be **accessed directly**by the **class name** and doesn’t need any object
* Syntax : <***class-name>.<method-name>***
* A static method cannot refer to “this” or “super” keywords in anyway
* A static method can be overloaded but not over ridden
* Constructors cannot be static.

**Ques - Create and overload constructors.**

**Ans-** Java Constructor overloading is a technique in which a class can have any number of constructors that differ in parameter list. The compiler differentiates these constructors by taking into account the number of parameters in the list and their type.

public class Test{

private String message;

public Test(){

message = "Hello World!";

}

public Test(String message){

this.message = message;

}

public String getMessage(){

return message ;

}

public void setMessage(String message){

this.message = message;

}

public static void main(String[] args) {

Test test = new Test();

System.out.println(test.getMessage());

Test test1 = new Test("Welcome");

System.out.println(test1.getMessage());

}

**Ques -Differentiate between default and user defined constructors  
Apply access modifiers.**

**Ans -**

## Default Constructor

* A default constructor is a **0 argument constructor**which contains a no-argument call to the super class constructor.
* To assign default values to the newly created objects is the main responsibility of default constructor.
* Compiler writes a default constructor in the code only if the program does not write any constructor in the class.
* The access modifier of default constructor is always the same as a class modifier but this rule is applicable only for **“public”**and **“default”** modifiers.

## Parameterized Constructors

* The parameterized constructors are the constructors having a **specific number of arguments** to be passed.
* The purpose of a parameterized constructor is to assign user-wanted specific values to the instance variables of different objects.
* A parameterized constructor is written explicitly by a programmer.
* The access modifier of default constructor is always the same as a class modifier but this rule is applicable only for **“public”** and **“default”** modifiers.

. The four access modifiers are –

## Default Access Modifier - No Keyword

Default access modifier means we do not explicitly declare an access modifier for a class, field, method, etc.

Eg- String version = "1.5.1";

boolean processOrder() {

return true;

## Private Access Modifier - Private

Methods, variables, and constructors that are declared private can only be accessed within the declared class itself.

Private access modifier is the most restrictive access level. Class and interfaces cannot be private.

**Eg-**

public class Logger {

private String format;

public String getFormat() {

return this.format;

}

public void setFormat(String format) {

this.format = format;

}

## Public Access Modifier - Public

A class, method, constructor, interface, etc. declared public can be accessed from any other class. Therefore, fields, methods, blocks declared inside a public class can be accessed from any class belonging to the Java Universe.

**Eg-**

public static void main(String[] arguments) {

// ...

}

## Protected Access Modifier - Protected

Variables, methods, and constructors, which are declared protected in a superclass can be accessed only by the subclasses in other package or any class within the package of the protected members' class.

The protected access modifier cannot be applied to class and interfaces. Methods, fields can be declared protected, however methods and fields in a interface cannot be declared protected.

**Eg-**

class AudioPlayer {

protected boolean openSpeaker(Speaker sp) {

// implementation details

}

}

class StreamingAudioPlayer extends AudioPlayer {

boolean openSpeaker(Speaker sp) {

// implementation details

}

}

**Ques-Apply encapsulation principles to a class**

**Ans -**Encapsulation is a mechanism of wrapping the data (variables) and code together as a single unit. In encapsulation, the variables of a class will be hidden from other classes, and can be accessed only through the methods of their current class.

**To achieve encapsulation in Java −**

* Declare the variables of a class as private.
* Provide public setter and getter methods to modify and view the variables values.

## Benefits of Encapsulation

* The fields of a class can be made read-only or write-only.
* A class can have total control over what is stored in its fields.
* 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.

**Ques- Determine the effect upon object references and primitive values when they are passed into methods that change the values.**

**Ans-** When primitive values are passed into methods, a copy is passed in. The copy of the primitive can changes value inside the method body, the original copy won't be effected.  
When an object reference is passed into method, a pointer to the reference is passed in. The object can be changed via the object reference regardless inside or outside of method body.

class test{

  public static void main(String...args) {

    test t = new test();

    int a = 1;

    t.passCopy(a);

    System.out.println(a);  //original a didn't change value

    String b = "String-outside";

    System.out.println(t.passCopy2(b));

    System.out.println(b);  //original b didn't change value

    StringBuilder sb = new StringBuilder("StringBuilder-outside");

    t.passRef(sb);

    System.out.println(sb);

  }

  void passCopy(int a) {

    a = 10;

  }

  String passCopy2(String b) {

    b = "String-inside";

    return b;

  }

  void passRef(StringBuilder sb) {

    sb.append("StringBuilder-inside");

  }

}

**Ques- Working with Inheritance  
Describe inheritance and its benefits**

**Ans-** [Inheritance](https://www.roseindia.net/java/language/inheritance.shtml) allows access of properties and methods of super class by a sub class. Extend keyword is used to inherit all the properties of the super-classes by subclasses.

**There are two types of inheritance in Java:**

1. **Simple Inheritance**
2. **Multilevel Inheritance**

**Simple Inheritance:**

Single inheritance or One-level inheritance is when a subclass is derived from parent class. There is only a sub class and it's parent class.

**Multilevel Inheritance:**

Multiple Inheritance is when a subclass is derived from a derived class. The derived class is called the subclass or child class for it's parent class and this parent class works as the child class for it's just above (parent) class.

**Multilevel inheritance can go up to any number of level.**

**Benefits of using inheritance:**

* A code can be used again and again
* Inheritance in Java enhances the properties of the class, which means that property of the parent class will automatically be inherited by the base class
* It can define more specialized classes by adding new details.

**Ques -Develop code that makes use of polymorphism;**

**Ans-** Polymorphism is one of the [OOPs](https://beginnersbook.com/2013/04/oops-concepts/) feature that allows us to perform a single action in different ways.

class Animal{

void eat(){System.out.println("eating...");}

}

class Dog extends Animal{

void bark(){System.out.println("barking...");}

}

class TestInheritance{

public static void main(String args[]){

Dog d=new Dog();

d.bark();

d.eat();

}

}

**Ques: Develop code that overrides methods.**

**Ans-**

If subclass (child class) has the same method as declared in the parent class, it is known as **method overriding in java**.

**Usage of Java Method Overriding**

* Method overriding is used to provide specific implementation of a method that is already provided by its super class.
* Method overriding is used for runtime polymorphism

**Eg-**

**class** Vehicle{

  //defining a method

**void** run(){System.out.println("Vehicle is running");}

}

//Creating a child class

**class** Bike2 **extends** Vehicle{

  //defining the same method as in the parent class

**void** run(){System.out.println("Bike is running safely");}

**public** **static** **void** main(String args[]){

  Bike2 obj = **new** Bike2();//creating object

  obj.run();//calling method

  }

**Ques: Differentiate between the type of a reference and the type of an object.**

**Ans:**

| **Parameter of Comparison** | **Object** | **Reference** |
| --- | --- | --- |
| Basic definition | It is the instance of a class and all the elements it contains are based on the blueprint of the class. | A simply memory reference that points to where the object is stored in memory slot. |
| Format for creation | There is a simple format for creation of an object: ClassName reference\_variable = new ClassName(with parameter); | The memory reference is created alongside object. It is used in the format for object creation and is given a variable name |
| Elements | It contains methods and variables based on the class. | It contains a sequence of bits that store the address of the object. |
| Mutable | Objects have states and behaviors that can be changed, i.e, the state of the object can be changed. | The reference variable value cannot be changed. It can only remain as the data type that it was declared as. |
| Virtual meaning | It is a real world entity that holds some form of memory or data. | It is nothing but a variable name, which has no real meaning. It is like the name of a person, that references that person. |

**Ques- Determine when casting is necessary**

**Ans -** Casting is the process of making a variable behaves as a variable of another type. If a class shares an IS-A or inheritance relationship with another class or interface, their variables can be cast to each other’s type. Some times the cast is allowed and some times that cast is not allowed. Because, some of the cast will not show any error at compile time, but will fail at run time.

**Eg :**

Public class Ocp {}

Public class Oca {

Public static void main(String[] args) {

Oca oa = new Oca();

Ocp op = (Ocp)oa; // DOES NOT COMPILE

}

}

* In this example, the classes Ocp and Oca are not related through any class hierarchy that the compiler is aware of. Therefore, the code will not compile.
* Even though two classes share a related hierarchy, that doesn’t mean an instance of one can automatically be cast to another.

**Ques- Use super and this to access objects and constructors.**

**Ans-** super keyword refers immediate super(parent) class objects.super keyword is only used in case of Inheritance. Whenever we create instance of class, parent class instance is also created. The parent class instance can be accessed by ‘ super’ keyword.

**Eg-**

|  |
| --- |
| class Person  {      Person()      {          System.out.println("Person class Constructor");      }  }  class Student extends Person  {      Student()      {          // invoke or call parent class constructor          super();          System.out.println("Student class Constructor");      }  }  class Test  {      public static void main(String[] args)      {          Student s = new Student();      }  } |

**This Keyword-** this is a reference to the *current object*, the object whose method or constructor is being called. We can refer to any member of the current object from within an instance method or a constructor by using this.

We can also use the this keyword to call another constructor in the same class. Doing so is called an *explicit constructor invocation*.

**Eg-**

public class Rectangle {

private int x, y;

private int width, height;

public Rectangle() {

**this(0, 0, 1, 1);**

}

public Rectangle(int width, int height) {

**this(0, 0, width, height);**

}

public Rectangle(int x, int y, int width, int height) {

this.x = x;

this.y = y;

this.width = width;

this.height = height;

}

}

**Ques -Use abstract classes and interfaces  
Handling Exceptions**

## Ans - When a class declare with keyword abstract it is known as abstract class in java .A abstract class has no object to be created. An abstract class has both abstract and non-abstract methods and also constructor and class data members.

## Eg-

### abstract class hero{   abstract void show(); } class cricketer extends hero{ void show() { System.out.println("SRK is my favourite hero"); } public static void main(String args[]) {  hero obj = new cricketer();  obj.show(); } }

**INTERFACE**: Interface in java like a abstract class but interface does not support contains constructor but it is also support Multiple constructor as well as Compare to Abstract class. Members of interface cannot be static .Members of Abstract class can be static.

Interface in Java is a blueprint of a class. It has static constants and abstract methods. The interface in Java is a mechanism to achieve abstraction. There can be only abstract methods in the Java interface, not method body. It is used to achieve abstraction and multiple inheritance in Java. Java Interface also represents the IS-A relationship

**Eg-**

interface Polytechnic{

void results();

}

class topper implements Polytechnic

{

public void results()

{

System.out.println("Results declare ");

}

public static void main(String args[]){

Polytechnic cse4th = topper();

object.results();

 }

}

**Ques - Differentiate among checked exceptions, unchecked exceptions, and Errors**

**Ans-**

**1)Checked Exception:** The classes which directly inherit Throwable class except RuntimeException and Error are known as checked exceptions e.g. IOException, SQLException etc. Checked exceptions are checked at compile-time.

### 2)Unchecked Exception:The classes which inherit RuntimeException are known as unchecked exceptions e.g. ArithmeticException, NullPointerException, ArrayIndexOutOfBoundsException etc. Unchecked exceptions are not checked at compile-time, but they are checked at runtime.

### 3)Error: Error is irrecoverable e.g. OutOfMemoryError, VirtualMachineError, AssertionError etc.

### Ques- Create a try-catch block and determine how exceptions alter normal program flow.

### Ans-getfile (1).jpg

### Ques: Describe the advantages of Exception handling.

**Ans- The advantages are:**

* **Separating Error-Handling Code from "Regular" Code**

Exceptions enable you to write the main flow of your code and to deal with the exceptional cases elsewhere.

* **Propagating Errors Up the Call Stack**

A second advantage of exceptions is the ability to propagate error reporting up the call stack of methods.

* **Grouping and Differentiating Error Types**

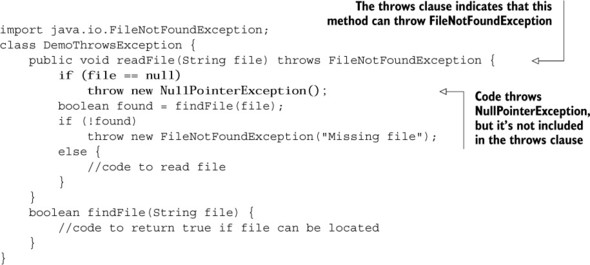
Because all exceptions thrown within a program are objects, the grouping or categorizing of exceptions is a natural outcome of the class hierarchy.

**Ques: Create and invoke a method that throws an exception  
Recognize common exception classes (such as NullPointerException, ArithmeticException, ArrayIndexOutOfBoundsException, ClassCastException)**

## Ans: Exception: An exception is an event, which occurs during the execution of a program,that disrupts the normal flow of the program’s instructions.

**Throwing Exception:**Creating an exception object and handing it to the runtime system is called throwing an exception.

**Handling Exception:**After a method throws an exception, the runtime system attempts to find something to handle it. The set of possible “somethings” to handle the exception is the ordered list of methods that had been called to get to the method where the error occurred. The list of methods is known as the call stack ().



There are given some scenarios where unchecked exceptions can occur. They are as follows:

**1) Scenario where ArithmeticException occurs**

If we divide any number by zero, there occurs an ArithmeticException.

1. int a=50/0;//ArithmeticException

**2) Scenario where NullPointerException occurs**

If we have null value in any variable, performing any operation by the variable occurs an NullPointerException.

1. String s=null;
2. System.out.println(s.length());//NullPointerException

**3) Scenario where NumberFormatException occurs**

The wrong formatting of any value, may occur NumberFormatException. Suppose we have a string variable that have characters, converting this variable into digit will occur NumberFormatException.

1. String s=“abc”;
2. int i=Integer.parseInt(s);//NumberFormatException

**4) Scenario where ArrayIndexOutOfBoundsException occurs**

If we are inserting any value in the wrong index, it would result ArrayIndexOutOfBoundsException as shown below:

1. int a[]=new int[5];
2. a[10]=50; //ArrayIndexOutOfBoundsException

**==========================END====================================**