**A Project Report ON**

**JAVA PROJECT**

**For**

**Certified Software Testing**

**BY**

**Mayuri Siddhu Bandgar**

**Batch No-7669**

**Under Guidance** **by:-Mohana Priya**

EDUBRIDGE

|  |  |
| --- | --- |
|  | INDEX |
| 1. | INTRODUCTION |
| 2. | Java Concept |
| 3. | Java Program |
| 4. | Output |
| 5. | Array |
| 6. | Exception handing |
|  |  |

**1.INTRODUCTION**

**Java-** **JAVA** was developed by James Gosling in the year **1995**,  It is a simple programming language. Java makes writing, compiling, and debugging programming easy. It helps to create reusable code and modular programs. [Java](https://www.geeksforgeeks.org/java/) is a class-based, object-oriented programming language and is designed to have as few implementation dependencies as possible. A general-purpose programming language made for developers to *write once run anywhere* that is compiled Java code can run on all platforms that support Java. Java applications are compiled to byte code that can run on any Java Virtual Machine. The syntax of Java is similar to c/c++.

**2.JAVA CONCEPT**

**2.1Java in all concept**

Object means a real-world entity such as a pen, chair, table, computer, watch, etc. **Object-Oriented Programming** is a methodology or paradigm to design a program using classes and objects. It simplifies software development and maintenance by providing some concepts:

* [Object](https://www.javatpoint.com/object-and-class-in-java)
* Class
* [Inheritance](https://www.javatpoint.com/inheritance-in-java)
* [Polymorphism](https://www.javatpoint.com/runtime-polymorphism-in-java)
* [Abstraction](https://www.javatpoint.com/abstract-class-in-java)
* [Encapsulation](https://www.javatpoint.com/encapsulation)

**2.1.1 Object-** The object is instance of class. Object is real word entity like pen chair,computer,watch etc .It can be Physical or logical

**2.1.2 Class-** Class is collection of multiple methods,variables,datatype,printing statement constructor and object is called class.

**2.2.3Inheritance-**When one Object Acquires all the properties and behaviours of a parent object is called inheritance.

Is-A relationship which also known as parent child relative.

There are five types of inheritance

a)Single inheritance

b)Multilevel inheritance

c)Multiple inheritance

d)Hybrid inheritance

e)Hierarchical inheritance

**2.2.4 Polymorphism-** If one task is performed in different ways it is known as polymorphism.

There are two types

a)Compile time polymorphism

b)Runtime polymorphism

**Method overloading-**

If a [class](https://www.javatpoint.com/object-and-class-in-java) has multiple methods having same name but different in parameters, it is known as **Method Overloading**.

**Method Overriding-**

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

**2.2.5 Abstraction-**Hiding internal details and showing functionality is Known as abstraction.

In use abstract class and interface.

**2.2.6 Encapsulation- Encapsulation in Java** is a process of wrapping code and data together into a single unit, for example, a capsule which is mixed of several medicines.

**2.2 Constructor-**Constructor is block of codes like the method it is called when an instance of the class is created.

2.2.1 Types of constructor

a)Default constructor(no argument constructor)

b)Parameterized constructor

Rules-Constructor name must be same as its class name

No return type

**2.3 This** **Keyword**-There can be a lot of usage of **Java this keyword**. In Java, this is a **reference variable** that refers to the current object.

**2.4 Super** **keyword**-The **super** keyword in Java is a reference variable which is used to refer immediate parent class object.

**2.5 Scanner**-Scanner class in Java is found in the java.util package. Java provides various ways to read input from the keyboard, the java.util.Scanner class is one of them.

**2.6 Final** **keyword**-

The **final keyword** in java is used to restrict the user. The java final keyword can be used in many context. Final can be:

1. variable
2. method
3. class

**2.7 Access** **modifier**-The access modifiers in Java specifies the accessibility or scope of a field, method, constructor, or class.

There are four types of Java access modifiers:

* + 1. **Private**: The access level of a private modifier is only within the class. It cannot be accessed from outside the class.
    2. **Default**: The access level of a default modifier is only within the package. It cannot be accessed from outside the package. If you do not specify any access level, it will be the default.

**2.7.3 Protected**: The access level of a protected modifier is within the package and outside the package through child class. If you do not make the child class, it cannot be accessed from outside the package.

**2.7.4 Public**: The access level of a public modifier is everywhere. It can be accessed from within the class, outside the class, within the package and outside the package.

**2.8 Non** **access** **modifier**-Non-access modifiers provide information about the characteristics of a class, method, or variable to the JVM. Seven types of Non-Access modifiers are present in Java.

They are-

1. **Static-**The static keyword means that the entity to which it is applied is available outside any particular instance of the class
2. **Final-**The **final**keyword indicates that the specific class cannot be extended or a method cannot be overridden.
3. **Abstract-**abstract keyword is used to declare a class as partially implemented means an object cannot be created directly from that class.
4. **Synchronized-**synchronized keyword prevents a block of code from executing by multiple threads at once. It is very important for some critical operations.
5. **Volatile-**The volatile keyword is used to make the class thread-safe. That means if a variable is declared as volatile, then that can be modified by multiple threads at the same time without any issues.
6. **Transient-**
7. **Native-**The native keyword may be applied to a method to indicate that the method is implemented in a language other than Java.

**2.9 Collection**-

It provides interfaces and classes to implement various data structure and algorithm

The collection in java is a framework that provides an architecture to store and manipulate the group of objects.

Java collection can achieve all the operations that you perform on a data such as searching ,sorting, insertion, manipulation, and deletion.

Java collection means a single unit of objects. Java collection framework pr ovides many interface(set,List,Queue,Deque)and classes.

The collection framework represents a unified architecture for storing and manipulating a group of obect.

1.interfaces and its implementation,i.e.classes

2.Algorithm

3.**JAVA** **PROGRAM**

**3.1 Java Program-**

**package** Project; //Create the package

**import** java.util.Scanner; //import Scanner the user of the given

**class** Driver1 {

String name; //define the variable

**int** mobile;

**public** Driver1() {

**this**.name = "Name of car"; //use this keyword

**this**.mobile = 123456;

}

**public** **void** drive() { //Print the statement

System.***out***.println("Drive started..." + "Have a nice drive!");

}

**public** **void** profile(String name) { //use this keyword by the name

**this**.name = name;

}

**public** **void** profile(String name, **int** mobile) { //

**this**.name = name;

**this**.mobile = mobile;

}

**public** String getName() { //

**return** name;

}

}

**class** Car {

Driver driver;

Beverages b;

String carChoice;

Car() {

**this**.carChoice = "SUV";

}

Car(String carChoice) {

**this**.carChoice = carChoice;

}

**void** GetInTheCar(Driver driver) {

System.***out***.println("Hey " + driver.getName()

+ " Enjoy driving with your " + **this**.carChoice + " Car");

driver.drive();

}

**void** EnjobyBeverages() {

System.***out***.println("Want Beverage?" + " Enter 1 for Tea/ 2 for Coffee!");

@SuppressWarnings("resource")

Scanner s = **new** Scanner(System.***in***);

**int** choice = s.nextInt();

**if** (choice == 1) { //use if statement

b = **new** Tea();

}

**if** (choice == 2) {

b = **new** Coffee();

}

b.getBeverage();

}

}

//

**abstract** **class** Beverages {

**private** **void** addHotWater() {

System.***out***.println("Adding hot water");

}

**private** **void** addMilk() {

System.***out***.println("Adding hot milk");

}

**private** **void** addSugar() {

System.***out***.println("Adding Sugar");

}

**public** **void** getMixture() {

System.***out***.println("Your Beverage is " + "getting ready...");

addHotWater();

addMilk();

addSugar();

}

**public** **abstract** **void** getBeverage(); //use abstract method

**public** **abstract** **void** addIngredients();

}

// create interface

**interface** Clean {

**void** cleanPot();

}

**class** Tea **extends** Beverages **implements** Clean {//create class extends the interface

**public** **void** addIngredients() {

System.***out***.println("Tea Bag added");

}

**public** **void** getBeverage() {

cleanPot();

getMixture();

addIngredients();

System.***out***.println("Tea's Ready! Enjoy");

}

**public** **void** cleanPot() {

System.***out***.println("Cleaning tea pot...");

}

}

**class** Coffee **extends** Beverages **implements** Clean { //create interface of the class

//Use This Override methods

**public** **void** addIngredients() {

System.***out***.println("Coffee Bag added");

}

**public** **void** getBeverage() {

cleanPot();

getMixture();

addIngredients();

System.***out***.println("Coffee's Ready! Enjoy");

}

**public** **void** cleanPot() {

System.***out***.println("Cleaning Coffee pot...");

}

}

**class** Driver{

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

System.***out***.println("Welcome for a nice Drive!");

@SuppressWarnings("resource")

Scanner s = **new** Scanner(System.***in***);

Driver peter = **new** Driver();

peter.profile("peter");

Car c;

System.***out***.println("Want to Choose Car? "+ "Press 0 for NO / 1 for YES");

**int** carType = s.nextInt();

**if** (carType == 1) {

System.***out***.println("Enter Car name");

String carName = s.next();

c = **new** Car(carName);

} **else** {

c = **new** Car();

}

c.GetInTheCar(peter);

c.EnjobyBeverages();

}

**public** String getName() {

**return** **null**;

}

**public** **void** drive() {

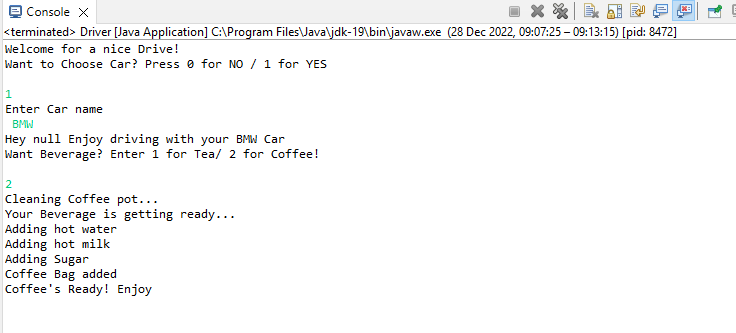
}

**private** **void** profile(String string) {

}

}

**OUTPUT -**



**3.2 Array**-

Array is a data structure consisting of a collection of elements (Values & variable)each identified by at least one array index or key

**Program-**

**public** **class** Checkequality {

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

**final** **int**[] array1 = { 11, 22, 33, 44, 55 };

**final** **int**[] array2 = { 11, 22, 33, 44, 55 };

**final** **boolean** intCheck = *equals*(array1, array2);

System.***out***.println("Two Integers are Equal :: " + intCheck);

}

**public** **static** **boolean** equals(**final** **int**[] a, **final** **int**[] a2) {

**if** (a == a2)

**return** **true**;

**if** (a == **null** || a2 == **null**)

**return** **false**;

**final** **int** length = a.length;

**if** (a2.length != length)

**return** **false**;

**for** (**int** i = 0; i < length; i++)

**if** (a[i] != a2[i])

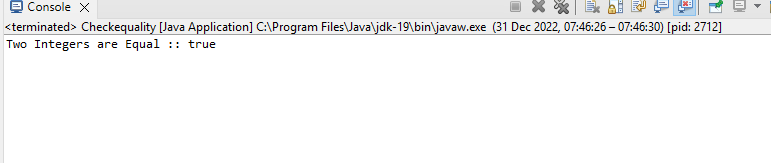
**return** **false**;

**return** **true**;

}

}

**Output-**

****

**3.3 Exception Handling-**

exception handling is a mechanism to handle runtime errors such as classnot found exception,IOexception,remote exception

Types of java Exception-

a)Checked Exception

b)Unchecked Exception

c)Error-

Java Exception keyword-

1)try-try keyword is used to specify block where we should place exception code

2)catch-catch keyword is used to handle the exception

3)finally-finally keyword is used to execute the important code of the program

4)Throw-throw keyword is used to throw an exception

5)Throws-throws keyword is used to declare an exception

**Program-**

**public** **class** finallytryblock {

**static** **void** run() {

**try** {

System.***out***.println("running");

**throw** **new** RuntimeException("demo");

} **finally** {

System.***out***.println("run finally");

}

}

// Return from within a try block.

**static** **void** play() {

**try** {

System.***out***.println("playing");

**return**;

} **finally** {

System.***out***.println("play finally");

}

}

// Execute a try block normally.

**static** **void** eat() {

**try** {

System.***out***.println("eatting");

} **finally** {

System.***out***.println("eat finally");

}

}

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

**try** {

*run*();

} **catch** (Exception e) {

System.***out***.println("Exception caught");

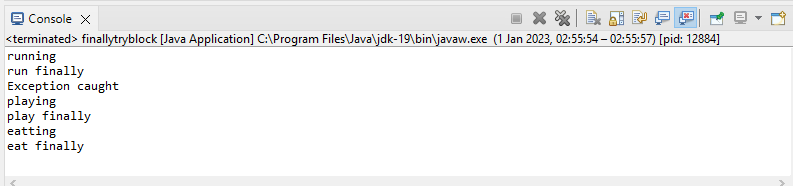
}

*play*();

*eat*();

}}

**OUTPUT-**

****