From,
Karthik Nair, 4EA, 00229802021

TO,
Dr. Neha Verma

Date 26/03/23

Java Programming Assignment 1

Q1. What are the various features of Java?

Java is a popular and widely-used programming language that has the following features that make it popular among developers:

Platform Independent: Java code can run on any operating system that has a Java Virtual Machine or JVM, installed, regardless of the underlying software and hardware architecture. The JVM acts as an interpreter between Java bytecode and the operating system.

Portable: Java code can be compiled into bytecode, which can be run on any platform that has a JVM installed. This makes Java code highly portable and easy to distribute.

Object-Oriented Programming: Java is an object-oriented programming language, which means it is based on the concepts of objects and classes. Object-oriented programming allows developers to create reusable code that is easier to manage and maintain.

Garbage Collection: Java has automatic garbage collection, which means that the system automatically frees up memory that is no longer needed by the application. This helps in preventing

vijeta

memory leaks and other memory related issues.

Multi-threading: Java supports multi-threading, which allows developers to create applications that can perform multiple tasks simultaneously. This helps to improve the performance of applications and allows them to be more responsive to user input.

Security: Java has built-in security features that help in preventing unauthorized access and protect against security threats. For example, Java's security manager allows developers to control the access of applications to system resources.

Rich APIs: Java has a large library of application Programming Interfaces (APIs) that provide developers with access to a wide range of functions and services. This makes it easier for developers to create applications that interact with other software systems and services.

Dynamic: Java is a dynamic programming language that allows for dynamic class loading and dynamic compilation of code. This allows developers to write code that can adapt to changing conditions and requirements.

Robust: Graceful errors and exception handling using built-in error handling mechanisms that helps in creating reliable and stable applications.

Date / /

Q2. Difference between final, finally and finalize

`final` : In java, `final` is a keyword that is used to declare a variable, method or class constant and unmodifiable.

- a variable or reference that is declared as `final` can not be changed.
- A method declared as `final` can not be overriden.
- A class declared as `final` can not be extended

```
public class final_usage {
    public static void main (String[] args) {
        final int a = 10;
        a = 20;
    }
}
```

- This program results in a compile-time error : the value of `a` can not be changed.

`finally` : In Java, `finally` is a block of that is used in exception handling to ensure that a block of code is always executed, whether or not an exception is thrown. The finally block is executed after the `try` and `catch` block have finished executing.

Date

```
try {
    // Some code that may throw an exception
} catch (Exception e) {
    // handle the exception
} finally {
    // Executes regardless of whether exception occured
}
```

'finalizer': In Java, 'finalizer' is a method that is called by the garbage collector before an object is destroyed. The 'finalize()' method can be overriden in a class to define custom cleanup actions that should be performed before the object is garbage collected. 'finalize()' should be avoided if possible, since it causes performance issues.

```
public class MyClass {
    private String name;

    public MyClass(String name) {
        this.name = name;
    }

    public void finalize() {
```

Date / /

System.out.println("Object" + name + "garbage collected");
}

// main
}

Q3. Write a short note on exception handling model.

In Java, the exception handling model is a mechanism that allows developers to handle runtime errors or exceptional situations in a structured and controlled way.

Java's exception handling model creates an exception object when an error occurs at runtime, including information about the error's type and location. The exception is thrown and passed up the call stack until it's caught by a try-catch block, which has code handle the exception.

```
public class ExceptionHandling {
    public static void main(String[] args) {
        int x = 4;
        int y = 0;
        try {
            int result = x / y;
        } catch (ArithmeticException e) {
            System.out.println("Caught" + e);
        }
    }
}
```

Date

Q4. Write a program on Constructor and method overloading.

```
public class OverloadingExample {
    private int value;
    // Constructor with no args
    public OverloadingExample() {
        this.value = 0;
    }

    // Constructor with one argument
    public OverloadingExample(int value) {
        this.value = value;
    }

    // Method with no arguments
    public void setValue() {
        this.value = 0;
    }

    // Method with one argument
    public void setValue(int value) {
        this.value = value;
    }

    public static void main(String[] args) {
        OverloadingExample Obj1
        = new OverloadingExample();

        OverloadingExample Obj2 = new OverloadingExample(10);
```

Date __/__/__

```
// Call methods with different arguments

Obj1.setValue();
Obj2.setValue(20);

System.out.println(Obj1.value);
System.out.println(Obj2.value);

    }
}

/* output
   0
   20
*/
```

Q5. Implement multiple inheritance with the help of a program.

- In Java, we use interfaces to implement multiple inheritance.

```
// Interface 1
interface Vehicle {
    void move();
}

// Interface 2
interface Animal {
    void eat();
}
```

```
// class implementing both interfaces
class Horse implements Vehicle, Animal {
    public void move () {
        System.out.println ("Horse is running..." );
    }

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

// main class
public class MultipleInheritanceExample {
    public static void main (String [] args) {
        Horse horse = new Horse ();
        horse.move ();
        horse.eat ();
    }
}

/* Output
Horse is running...
Horse is eating...
*/
```

Q6- What is 'super' keyword? Explain in detail with the help of a program.

Date __/__/__

The 'super' keyword is used to access members (members, fields and constructors) of a superclass from within a ~~Stubla~~ subclass.

This keyword is useful when we want to override a method or use a field from superclass in subclass.

```
// Superclass
class Animal {
    protected String name;

    public Animal(String name) {
        this.name = name;
    }

    public void eat() {
        System.out.println("The animal is eating");
    }
}

// Subclass
class Dog extends Animal {
    private String breed;

    public Dog(String name, String breed) {
        super(name); // call Superclass constructor
        this.breed = breed;
    }
```

Date

```
    public void bark() {
        System.out.println
(System.out.println

        System.out.println("The " + breed + "dog named "
            + name + " is barking");
    }
    // override superclass method
    // override superclass method

    public void eat() {
        super.eat(); // call superclass method
        System.out.println("The" + breed + "dog named"
            + name + "is also eating");
    }
}

// main class
public class SuperKeywordExample {

    public static void main(String[] args) {

        Dog dog = new Dog("Buddy",
            "Labrador Retriever");

        dog.bark();
        dog.eat();

    }
}
```

Date / /

/* Output

The Labrador Retriever dog named Buddy is barking.

The animal is eating.

The Labrador Retriever dog named Buddy is also eating.

/*

In the 'Dog' constructor, we use the 'super' keyword to call the 'Animal' constructor and initialize the 'name' field. In the 'Dog' 'eat()' method, we use the 'super' keyword to call the 'Animal' 'eat()' method before printing a message specific to the 'Dog' class.

In 'main', we create a 'Dog' object and call its 'bark()' and 'eat()' methods. The 'bark()' method prints a message using both the 'breed' and 'name' fields of the 'Dog' object, while the 'eat()' method first calls the 'Animal' 'eat()' method using the 'super' keyword, and then prints a message using both the 'breed' and 'name' fields of the 'Dog' object.

Date

Q7. Write a program on dynamic method dispatch.

```
// Superclass
class Animal {
    public void makeSound () {
        System.out.println ("The animal makes a sound!");
    }
}

// Subclass
class Cat extends Animal {
    @override
    public void makeSound () {
        System.out.println (" Cat meows. ");
    }
}

// Subclass

class Dog extends Animal {

    @override
    public void makeSound () {
        System.out.println (" Dog barks. ");
    }
}

// main class

public class DynamicMethodDispatchExample {
    public static void main (String [] args) {
```

Date ___/___/___

```
Animal animal1 = new Animal();
Animal animal2 = new Cat();
Animal animal3 = new Dog();

animal1.makeSound();
animal2.makeSound();
animal3.makeSound();
    }
}
```

/* Output

The animal maker a sound.
Dog ~~ba~~
Cat meows.
Dog barks.
*/

Here, since the 'makeSound()' method is overriden in the 'Cat' and 'Dog' subclasses, Java uses dynamic method dispatch to determine the actual method to call on runtime.

Q8. What are checked and ~~unchecked~~ unchecked exception?

Checked exceptions

- Exceptions that the compiler requires to be caught or declared in the method signature using the 'throws'

Date / /

Keyword.

- Used to handle errors that can occur during normal program execution, such as I/O errors, network connection errors, database access errors.

- Checked exceptions are recoverable, i.e the program can recover from the error by handling the exception in a try-catch, or by declaring the Exception in the method signature and allowing the caller to handle it.

- Examples include `IOException`, `SQLException`, `ClassNotFoundException`.

Unchecked Exceptions

- Unchecked exceptions aren't required to be caught or declared in the method signature. They are typically used to handle programming errors that occur at runtime, such as null pointer exceptions, division by zero, and array index out of bounds.

- These exceptions are fatal, i.e unrecoverable and terminate the program.

- e.g `NullPointerException`, `ArithmeticException`

vijeta

Date / /

Q. What is the difference between 'throw' and 'throws'?

In Java, 'throw' and 'throws' are two keywords used in exception handling.

* throw

- used to explicitly throw an exception in a method.

- when you use 'throw', you indicate that something has gone wrong in your method and you want to create an exception object and throw it back to the caller.

```
public int Square (int num) {
    if (num < 0) {
        throw new IllegalArgumentException
        ("Input must be non-negative");
    }
    return num * num;
}
```

* throws

- used in the method signature to indicate that the method can potentially throw a checked exception.

- when you use 'throws', you tell the

Date / /

compiler that your method may throw an exception, and the compiler will ensure that the exception is either caught or declared in the calling method.

```
public void readFile(String filename) throws
    IOException {  //Code to read file
    }

// 'readFile' method might throw 'IOException'
```

Q10. Explain How Java implements call by reference using an example.

Call by reference can be implemented in Java by passing a copy of a reference object to its class itself.

```
class Test {
    int a, b;
    Test (int i, int j) {
        a = i;
        b = j;
    }

    void multi (Test ob)
    {
        ob.a *= 2;
        ob.b *= 2;
    }
}
```

//vijeta//

Date / /

```
public class CallByReference {

    public static void main (String [] args) {
        Test ob = new Test (15, 20);

        System.out.println ("BEFORE: " + ob.a
        + "  " + ob.b );

        ob.math (ob );

        System.out.println ("AFTER: " + ob.a
        + "  " + ob.b );

    }
}
```

/* Output

Before : 15 20
After : 30 10

/*

Technically, Java is still implementing call by value since we are actually passing a copy of the reference to the object, not the object itself to the method.

Karthik Nair
0022980202|

Date / /

Q11. What is abstract class?

- When a superclass is unable to create a meaningful implementation for a method, then it is important to define the method as ABSTRACT.

- This means superclass defines generalized form that will be shared by all of its subclasses leaving it to each subclass to fill its details.

- If a class has even one abstract class method, then the class becomes abstract.

```
abstract class A {
        abstract void callme();
        void callmetoo(){
            System.out.println("Simple method.");
        }
    }

class B extends A {
        void callme() {
            System.out.println("B's Callme");
        }
    }
class Demo{ public static void main(String[] args){
    B b1 = new B();
    b1.callme();
    b1.callmetoo(); }}
```

output
B's Callme
Simple method