Wipro Primer Pre Skilling Assignment 4
Practice Assignment

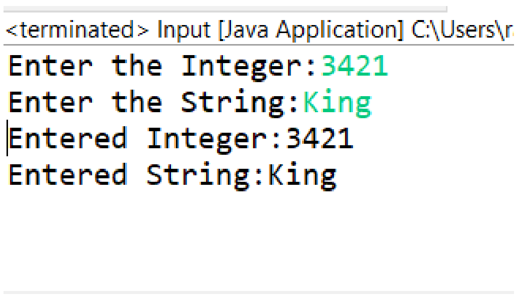
1. Explain how you would use the Scanner class to read user input from the console. Provide an example of code that reads an integer and a string from the user.

- Scanner Class comes under **java.util** package. First, we need to import the package which allows the user to read values of various types in Java.
- Then we need to create the object for the Scanner class with the help of new keyword. it have constructor with parameter **System.in** which is used to take inputs from standard input.
- Then with the help of **nextInt()** in-built method, which is used to read Integers and with the help of **next()** or **nextLine()** in-built method, which is used to read Strings.

Example program:

```
package com.WiproLmsAssignment4;
import java.util.*;
public class Input {
    public static void main(String[] args){
            Scanner sc=new Scanner(System.in);
            System.out.print("Enter the Integer:");
            int a=sc.nextInt();
            System.out.print("Enter the String:");
            String b=sc.next();
            System.out.println("Entered Integer:"+a);
            System.out.println("Entered String:"+b);
            }
    }
```

Output:



```
<terminated> Input [Java Application] C:\Users\r
Enter the Integer:3421
Enter the String:King
Entered Integer:3421
Entered String:King
```

2. Explain the use case for a static block in Java with an example.

Static block:

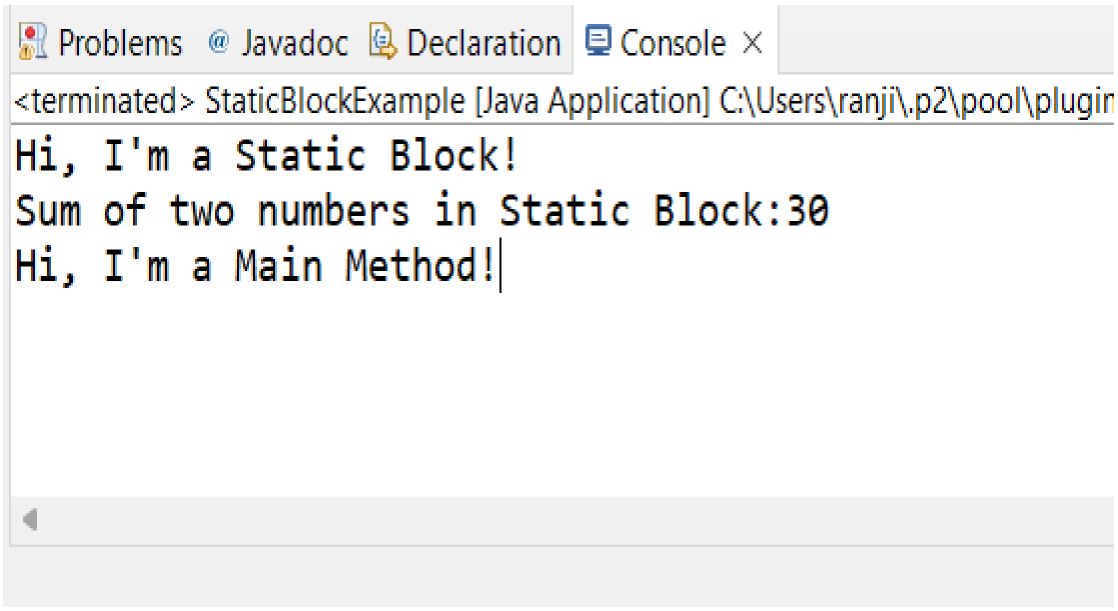
- Java supports a special block, called a static block (also called static clause).
- A static block in java is a code associated with static keyword, executed only once when the class is loaded into memory by the java classloader>
- It is used for static initialization of a class, ensuring that certain fields or operations are performed just once, typically at the beginning of the programs execution.

Example program:

```
package com.WiproLmsAssignment4;
public class StaticBlockExample {
        static{
                System.out.println("Hi, I'm a Static Block!");
                int a=10;
                int b=20;
                System.out.println("Sum of two numbers in Static Block:"+(a+b));
                }
        public static void main(String[] args){
                System.out.println("Hi, I'm a Main Method!");

                }
}
```

Output:



```
<terminated> StaticBlockExample [Java Application] C:\Users\ranji\.p2\pool\plugir
Hi, I'm a Static Block!
Sum of two numbers in Static Block:30
Hi, I'm a Main Method!
```

3. Explain the concept of a static nested class and its use cases.

Static Nested Class:

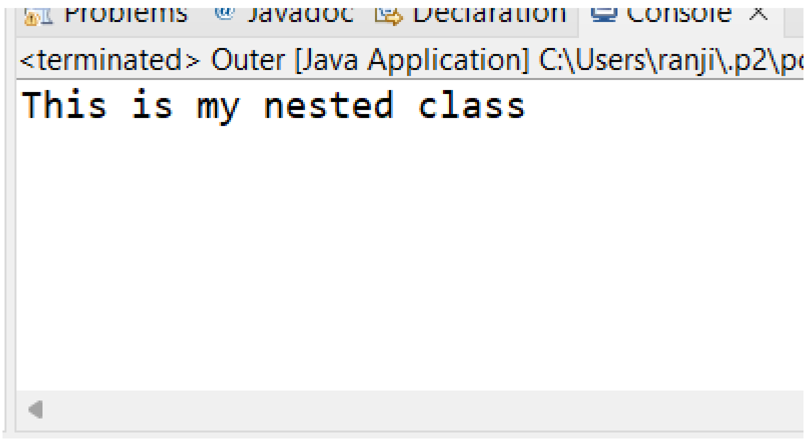
A static inner class is a nested class which is a static member of the outer class. It can be accessed without instantiating the outer class, using other static members. Just like static members, a static nested class does not have access to the instance variables and methods of the outer class.

Example:

```
package com.WiproLmsAssignment4;
public class Outer {
        static class NestedDemo {
          public void display() {
            System.out.println("This is my nested class");
          }
        }

        public static void main(String args[]) {
          Outer.NestedDemo nested = new Outer.NestedDemo();
          nested.display();
        }
      }
```

Output:



```
<terminated> Outer [Java Application] C:\Users\ranji\.p2\p
This is my nested class
```

Use Cases static nested class:

1. Grouping Related Classes: Static nested classes are useful for grouping classes that are closely related to the outer class. It helps in organizing the code and improves code readability.

2. Encapsulation: If a class is only relevant to the outer class and does not need access to its instance variables or methods, it can be made static nested to encapsulate it within the outer class.

3. Improved Accessibility: Static nested classes have access to all static members (variables and methods) of the outer class, even if they are private. This allows for better encapsulation and access control.

4. Helper Classes: Static nested classes can be used as helper classes to perform tasks related to the outer class without cluttering the namespace with additional top-level class names.

4. Describe the importance of keywords in controlling program flow, flow control keywords including loops, conditional statements, and branching. Provide examples of how are employed in Java.

Flow controls decision making

- Decision-making is an important aspect of programming as it allows the execution of specific blocks of code based on certain conditions. In Java, a number of things facilitate decision-making, like if-else statements, switches, and ternary operators.
- The decision-making process in the programming is similar to that of real life. Programming is faced with situations where we would like to execute a given code block if the conditions are met.
- Programming languages use control statements to regulate the performance of a program in accordance with certain conditions. They are used to create a flow of execution based upon changes in the program's state, leading to an advance and branch.

Loops:

For: If the number of iterations is known in advance, then a "for loop" will be applied. It comprises three parts: initialization, condition, and increment/decrement. The loop initializes a variable, checks the condition, and executes code blocks for as long as this condition remains false

Example:

```
for(int i=0;i<n;i++) {
    System.out.println("Number of elements:"+i);
}
```

While: If it is unknown in advance what number of iterations are used, the while loop will be applied. Once the specified condition is met, it will constantly execute a code block. Before each increment, the condition shall be verified.

Example:

```
 int count = 0;
 while (count < 5) {
System.out.println("Number of element:"+count);
count++;}
```

do-while: The do-while loop is similar to the while loop, but the only difference is it checks the condition after each iteration. It ensures that the code block will be implemented at least once, regardless of whether this condition was initially incorrect.

Example:

```
int num=1;
do {
System.out.println("Number:"+num);
num++;

}while(num<=5);
```

For each loop: In Java programming, the for-each loop, also known as the enhanced for loop, is a convenient iteration construct that simplifies the process of iterating over arrays or collections. It does not require explicit indexing or manual iteration and makes it easy for users to access and apply each sequence element.

Syntax:

```
for (elementType element : arrayOrCollection) {

    // Code to be executed for each element

}
```

Example:

```
for(String:names) {
    System.out.println("Elements of array:"+names);
}
```

Conditional Statements:

if else: If the statement does tell us that if a condition is true, an order of statements must be executed in case it is false. What if we're going to do something else when the condition isn't true? That's the other statement we got here. we can use the else statement with the if statement to execute a code block when the condition is false.

Example:

```
if(n%2==0) {
    System.out.println("The entered number is even:"+n);
}
else {
    System.out.println("The entered number is odd:"+n);
}
```

If else if ladder:

- The IfElseIf ladder is a control flow in Java that allows programs to perform an evaluation of several conditions, executing different blocks of code on the basis of those conditions. It allows for forming a series of if-else statements in an orderly fashion.
- The IfelseElseIf ladder consists of multiple ifelseElseIf statements, where each ifelseElseIf condition is evaluated only if the preceding conditions are false. The ladder structure helps to organize and prioritize the conditions that need to be checked.

Example:

```
if(n==0){
        System.out.println(n+" is zero");
}
else if(n>0){
        System.out.println(n+" is positive");
}
else{
        System.out.println(n+" is negative");
}
```

switch case: It will will evaluate an expression against multiple possible cases and execute one or more blocks of code based on matching cases.

Example:

```
switch(ch){
case '+':
        System.out.println("I am Block 1 Addition");
  break;
case '-':
        System.out.println("I am Block 2 Substraction");
  break;
default:
        System.out.println("I am Default Block Multiplication");
}
```

Branching:
break: The break statement controls the flow of loop and switch statements to allow you to leave these blocks earlier than is necessary when some conditions are met. It provides an efficient tool to control the operation of a program and achieve desired behavior by means of specific criteria.

Example:
```
for (int i = 1;i<=10;i++) {
   if (i==5) {
      break; //Exit the loop when i is equal to 5
   }
   System.out.println("Number of elements:"+i);
}
```

continue: In Java, the continue statement is a control flow statement used within loops to skip the remaining code in the current iteration and proceed to the next iteration. On the basis of certain special conditions, it is common to use selective bypass for some parts of the loop body.

Example:
```
for (int i = 1;i<=10;i++) {
   if (i==5) {
      continue; //Skip the remaining code for i=5
   }
   System.out.println("Number of elements:"+i);
}
```

5.Explain the role of the super keyword in Java. How is it used to access members of a superclass, and can it be used in constructors? Provide an example.

Super Keyword:

The super keyword refers to superclass (parent) objects. It is used to call superclass methods, and to access the superclass constructor. The most common use of the super keyword is to eliminate the confusion between superclasses and subclasses that have methods with the same name.

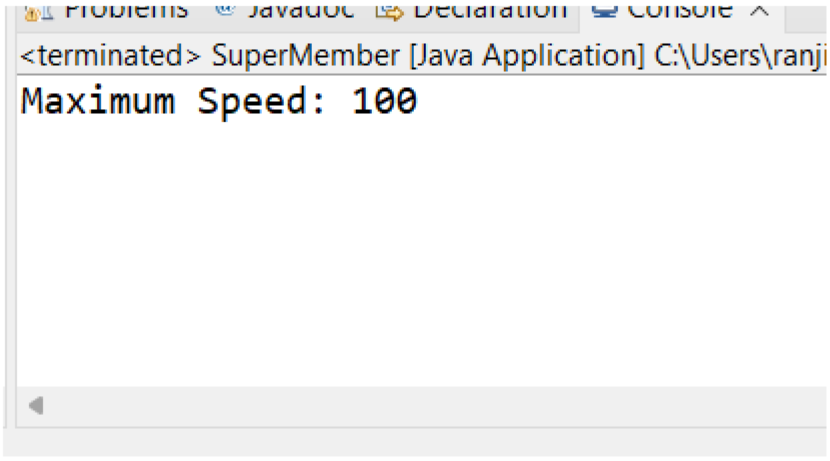
Accessing members of a superclass: This scenario occurs when a derived class and base class have the same data members.

Example:

```
package com.WiproLmsAssignment4;
class Vehicle { //Base class vehicle
	int maxSpeed = 100;
}
class Car extends Vehicle { //sub class Car extending vehicle
	int maxSpeed = 150;
	void display()
	{   // print maxSpeed of base class (vehicle)
		System.out.println("Maximum Speed: "
				   + super.maxSpeed);
	}
}
```

```java
class SuperMember{
	public static void main(String[] args)
	{
		Car small = new Car();
		small.display();
	}
}
```

Output:



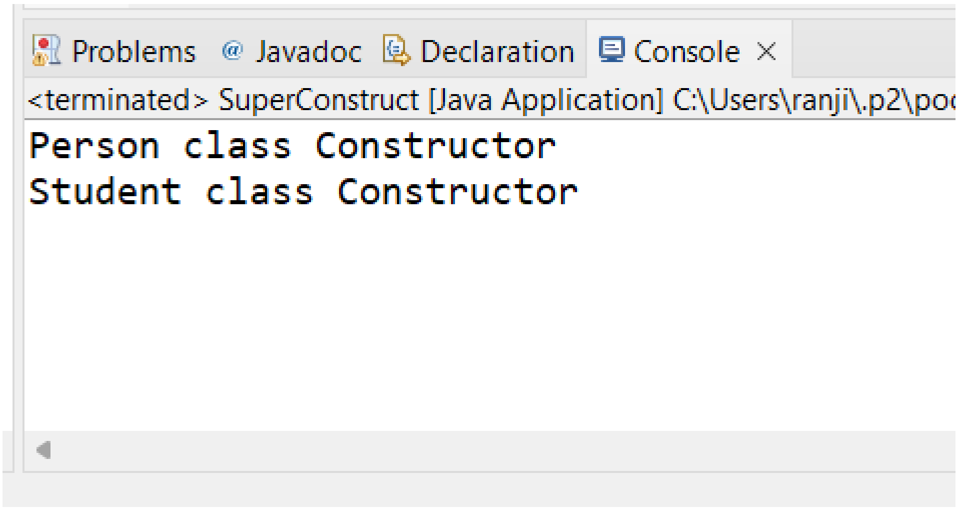
```
<terminated> SuperMember [Java Application] C:\Users\ranji
Maximum Speed: 100
```

Use of superclass with constructor: The super keyword can also be used to access the parent class constructor. One more important thing is that 'super' can call both parametric as well as non-parametric constructors depending on the situation.

Example

```java
package com.WiproLmsAssignment4;
class Person { //superclass Person
	Person()
	{
		System.out.println("Person class Constructor");
	}
}
class Student extends Person { //subclass Student extending the Person class
	Student()
	{
		super(); // invoke or call parent class constructor
		System.out.println("Student class Constructor");
	}
}
public class SuperConstruct {
	public static void main(String[] args)
	{
		Student s = new Student();
	}
}
```

Output:



```
<terminated> SuperConstruct [Java Application] C:\Users\ranji\.p2\poo
Person class Constructor
Student class Constructor
```

6. Discuss the significance of this keyword in Java. How does it differ from super, and when would you use it to refer to class members or constructors within the same class?

this keyword:

The this keyword refers to the current object in a method or constructor. The most common use of the this keyword is to eliminate the confusion between class attributes and parameters with the same (because a class attribute is shadowed by a method or constructor parameter).

Difference between this and super keywords:

this keyword	super keyword
this is an implicit reference variable keyword used to represent the current class.	super is an implicit reference variable keyword used to represent the immediate parent class.
this is to invoke methods of the current class.	super is used to invoke methods of the immediate parent class.
this is used to invoke a constructor of the current class.	super is used to invoke a constructor of the immediate parent class.
this refers to the instance and static variables of the current class.	super refers to the instance and static variables of the immediate parent class.
this can be used to return and pass as an argument in the context of a current class object.	super can be used to return and pass as an argument in the context of an immediate parent class object.

when would we use it to refer to class members or constructors within the same class

Using this with constructor: From within a constructor, you can also use the this keyword to call another constructor in the same class. Doing so is called an explicit constructor invocation. Here's another Rectangle class, with a different implementation from the one in the objects section.

Example:

```
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;
    }
}
```

Explanation:

- This class contains a set of constructors. Each constructor initializes some or all of the rectangle's member variables. The constructors provide a default value for any member variable whose initial value is not provided by an argument. For example, the no-argument constructor creates a 1x1 Rectangle at coordinates 0,0. The two-argument constructor calls the four-argument constructor, passing in the width and height but always using the 0,0 coordinates. As before, the compiler determines which constructor to call, based on the number and the type of arguments.
- If present, the invocation of another constructor must be the first line in the constructor.