Assignment 4.

Date: / /

Q. What does the static keyword mean in Java? Explain static and non-static methods.

In Java the static keyword is used to indicate that a particular member (variable, method, block or nested class). belongs to the class itself.

static members in java -

1. static variables -
these are class level variables that are shared among all instances of class.

2. static methods -
methods declared with static keyword can be called without creating an instance of the class.
they can only access static variables or other static methods directly because they do not have access to instance variables or instance methods.

3. static Blocks -
Blocks of code that are executed when the class is loaded into memory.

4. static Nested class -
A nested class that does not require an instance of the outer class to be instantiated.

Date: / /

Static Vs Non-Static

```
eg- class A {
      static int b = 10; // static var
      int c        = 20;   // Non static var.

    static void method()
      {
      System.out.println("Static method");
      Sysoln (b);
      Sysoln(c); // can not access non-static
                     variable directly.
      }

    void method1{ ) {
      Sysoln ("Non-static method");
      // can access both static & non-static
         variables.
      Sysoln(b);
      Sysoln (c);
      }
    }

  public class Test {
      public static void main (String args[]
      {
        A. method (); // calling static
                                   method
        A. a = new A (); // calling non-sta
           a. method1 ();     method require
                                 an object.
```

Date: / /

Q. What is the role of static keyword in the context of memory management.

1. Memory allocation in the method area -
when a class is loaded into memory by the Java virtual machine (JVM), all static members of the class are stored in a special area of memory called the method Area.

2. Single copy per class -
static members have only a single copy that is shared by all instances of the class.

3. Lifecycle of static members -
static members exist as long as the class is loaded in the JVM.
they are created when the class is loaded and are destroyed when the class is unloaded.

4. Memory consistency Issues -
since static variables are shared among all instances, improper use of static variables, especially in multithreaded application.

The efficient use of memory and accessibility without creating instances illustrates the role of static in memory management.

Date: / /

Q. Can static methods be overloaded and overridden in Java.

1. Overloading static methods -

overloading - multiple methods with same name but different parameters with in the same class.

static methods can be overloaded in Java just like instance methods.
overloading is resolved at compile-time based on the method-signature.

ex -

```
class A {
    static void display (int num)
    {
     Sysoln ("static method with int: "+num);
    }

    static void display (string str)
     {
      sysoln(" static method with
                string: "+ str);
      }
    }.
```

2- overriding static methods-

overriding - same method name with same parameter with different class.

ex-

```
class parent{
    static void show(){
        sysoln("static method in
                parent");
    }
}

class child extends Parent{
    static void show(){
        sysoln("static method in
                child");
    }
}
public class Test{
 public static void main(String args[])
 {
    parent.show();
    child.show();

    parent p = new child();
     p.show();

  }
 }.
```

Q.3 Significance of this 'final' keyword in java.

The final keyword in java is used to indicate that a variable, method, or class can not be modified after it is declared.

-> final Variable -

When a variable is declared as final, its value can not be changed once it has been initialized,

ex- final int MAX_VALUE =100;

-> final method -

A method declared as final. can not be overridden by subclasses, This is useful when you want to prevent subclasses from changing the implementation of a method.

```
ex- public class A {
        public final void display (){
           System.out.println("This is final
                                method");

        }
    }
    public class Derivedclass extends A{

          // code
    }.
```

→ Final class -
A class declared as final can not be extended by other classes. This is useful when you want to prevent a class from being subclassed.

```
public final class A{
        //code.
    }
```

Q.4. What are narrowing and widening conversions in java?

-

→ Widening conversion -
A widening conversion occurs when a smaller data type is converted into larger data type. this type of conversion is automatic because it is safe and there is no risk of data loss.

→ common widening conversions include-
① byte to short, int, long, float or double.

② short to int, long, float, double.

③ int to long, float or double

④ char to int, long, float, double.

```
ex- int a =100;
    long b = a ; // automatic widening
                    conversion from int
                    to long.
```

→ Narrowing Conversion -
A narrowing conversion occurs when a larger data type is converted into a smaller data type. this conversion is not automatic and can cause data loss, so it must be done using explicitly casting.

Common narrowing conversions include-
① double to float, long, int, short, byte or char.

② float to long, int, short, byte or char

③ long to int, short, byte or char.

```
ex- double a = 9.78;
    int b = (int) a ; // narrowing conversion
                        with explicit casting.
```

Narrowing conversions are risky because of they can lead to loss or overflow/ underflow issues.

Q.5 Examples of narrowing and widening conversions between primitive data types.

-) widening conversions -

ex - int to long.

```
int a =50;
long b = a; // int to long.
sysoln(b); //o/p - 50
```

ex - float to double.

```
float a= 50.2F;
double b = a; // float to double
sysoln(b); //o/p - 50.2
```

ex- char to int.

```
char a = 'A';
int b = a; //char to int.
sysoln(b);     // o/p = 65
```

-) Narrowing conversions

ex - double to int.

```
double a = 9.99;
int b =(int) a ; // double to int
sysoln(b); //o/p - 9 (fractional part lost).
```

ex.2. long to short.

```
long a = 100000;
short b = (short) a ; // explicit casting
sysoln(b);                long to short
                //o/p - 100000 (no data loss).
```

```
ex - int ~~a~~ = 130;
     byte b = (byte) a ; // explicit cast
                            int to byte
     sysoln(b);    // o/p = -126
                   (data loss due to
                        overflow).
```

(byte size = 8 bits (1 byte)
Range = -128 to 127)

ex - long to int

```
long l = 2147242625L;
int i = (int) l ;
      // narrowing conversion
         from long to int.
```

Date: / /

Q.6 How, Java handles potential Loss of precision during narrowing conversion.

Java handles potential loss of precision during narrowing conversions by requiring an explicit cast.
In this case we accept the risk of data loss or precision reduction.

ex - double to int ,

```
double d = 9.999;
int i = (int) d ;

        // o/p - fractional part is loss
              o/p = 9
```

similarly when narrowing from int to byte (larger to smaller), the cast forces the conversion despite the risk of overflow.

```
int i = 130;
byte b = (byte) i ;

        // o/p = -126
              due to overflow.
```

Date: / /

Q.7 Concept of automic widening conversion in java.

Automic widening conversion in java refers to the implicit (automatic) conversion of smaller primitive data type to a larger primitive data type. without requiring any explicit casting by the programmer.
this conversion is safe because of conversion betn ~~higher to~~ smaller to higher data types, ensuring to no data loss, .

widening conversion are typically from a type with a smaller range to a type with a larger range.

-> byte → short → int → long → float → double

-> char → int → long → float → double.

```
ex - int i = 100;
     long l = i; // automic widening.

ex - int i = 5;
     double d = 4.5;
     double result = i + d;

        // i is automatically widened
              to double.
```

Date: / /

Q.8. What are the implications of narrowing and widening conversions on type compatibility and data loss ?

→ Implications of widening convesions -

→ Type compatibility -
widening conversions are safe and maintain type compatibility.

→ No data loss

→ Easy of use.

→ Implications of Narrowing convessions -

→ potential data loss -
Narrowing conversion can lead to data loss or overflow or underflow.

→ Explicit cast required -
to perform narrowing convesions explicit cast is required, this serves as a warning that data loss or precision reduction might occur.