PAGE No.
DATE / /

Assignment - 4

1. What does the static keyword mean in Java? Explain the DB. static & non-static methods.

Ans - The static keyword in java has several important uses. It is primarily used to define class-level fields (variables) & methods, meaning they are shared among all instances of the class rather than being specific to a particular instance.

- Static Fields (Variables): When a field is declared static, it belongs to the class itself rather than to instance of the class. All instances of the class share the same static field.
- Static Methods: Static methods belong to the class, not to instance of the class. They can be called without creating an instance of the class. Static methods can only access static variables & other static methods directly.

Aspect	Static Methods	Non-static Methods
Belongs to	Class	Instance
Invocation	Called using class name.	Called using an object / instance of class
Access to instance variable	Cannot directly access non-static (instance) variable or methods (must create object/ instance to access)	Can directly access both static & non-static variables & methods.
Overriding	Cannot be overridden.	Can be overridden by subclass
Memory	Load once into memory when class is loaded.	Each object gets its own copy of non-static members.

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

Ans In the context of memory management, the 'static' keyword ensures that a variable or method is allocated memory only one throught the entire program lifecycle, regardless of how many instance of the class are created, effectively sharing the same memory space across all instances, leading to more efficient memory utilization.

1) Memory allocation in class are (Method Area)

Memory allocated once in meathod area (part of the heap where class-level data is stored) when class is loaded by JVM. Shared by all instances of class, & there is only one copy of static field in memory.

2) Lifetime of Static Members

Static field & methods exist as long as class is loaded into memory, even if no instance of class exist. They are loaded once during class loading & remain until the class is unloaded by JVM.

3) Reduced Memory Footprint

Shared among all instances of a class, it avoids creating duplicate copies for each object.

4) Class Level Access

We can access a static variable & method directly using class name, without needing an object instance.

5) Garbage Collection

Static members are not subject to GC as long as class is loaded. Unlike instance fields, which are eligible for GC when an object is no longer referenced, static fields remain in memory until the class is unloaded.

3. Can static methods be overloaded & overridden in Java? How static variable shared across multiple instance of a class?

Ans - Yes static methods can be overloaded in Java. Method overloading refers to having multiple methods with same name but different parameter lists within the same class. Since overloading is resolved at compile time, it works perfectly fine with static methods.

- No, static methods cannot be overridden in Java in the traditional sense of method overriding. In Java method overriding happens when a subclass provides a specific implementation for a method already present in superclass. However, static methods ~~are~~ are associated with class, so they cannot exhibit runtime polymorphism or dynamic binding which, which is essential for method overriding.

Static methods are resolved at compile, time, while instance method are resolved at runtime. Therefore, when static method is re-declared in a subclass, it does not override the superclass method but rather hides it.

This is called Method Hiding, not overriding

eg. show() → Parent
show() → Child
Parent p = new child()
p.show(); // Parent Show()

Static Variables are shared across multiple instances of a class

- Static variable are class-level variables, meaning they are shared across all instances of a class. They are not tied to any specific object but belongs to class itself.
- Static variables are allocated memory once when class is loaded into memory & all instances of the class share this same memory location.
- This means one instance of class modifies value of static variable, the change will be reflected to all instances of that class.

3. What is significance of the final keyword in Java?

Ans The final keyword in Java is used to apply restrictions on classes, methods or variables. It serves different purposes depending on where it used:

1) Final Variables

A final variable acts as a constant, meaning that its value cannot be changed once it is initialized. The variable must be initialized when it is declared or in the constructor.

Significance: It is used when you want to prevent subclasses from changing the implementation of a method. It is especially useful for security, consistency or performance reasons (eg. to avoid ~~or~~ overriding critical methods).

2) Final Methods

A final method cannot be overridden by subclasses.

Significance: It ensures that value of variable cannot be modified after initialization, which is useful for defining constants & for thread safety in concurrent programming.

3) Final Classes

A final class cannot be extended (inherited)

Significance: Prevents inheritance of the class. This is useful when you want to create immutable classes or classes whose functionality should not be altered by subclass (eg. the String class in Java is null)

4) Final Parameters

A final ~~parameter~~ keyword can also be applied to method parameters to prevent them from being modified inside the method.

Significance: Ensures that the value passed to the parameter is not accidentally modified, which can be

useful in ~~remaining~~ maintaining consistency & avoiding side effects.

4. What are narrowing & widening conversions in Java?

Ans In Java, narrowing & widening conversions refer to the process of converting one type into another. The conversion mainly occur between primitive data types. ~~&~~

1) Widening Conversion

- Widening conversion is process of converting a smaller or narrower data type into a larger or wider data type. This type of conversion happens automatically (implicitly) because there is no risk of data loss.
- Java automatically performs widening conversions as the smaller data type can be safely converted into a larger one without losing any information.

i.e. Automatic (Implicit) Conversion.

byte → short → int → long → float → double

char → int → long → float → double

```
eg. int no = 10;
    long no1 = no;   // 10
    float no2 = no;  // 10.0
```

2) Narrowing Conversion

- Narrowing conversion is the process of converting larger data type into smaller data type. This type of conversion must be explicitely performed by the programmer because there is a risk

6 - 7 Decimal Digits ⇒ Float

15 Decimal Digits ⇒ Double

of data loss or overflow. Narrowing is also called explicit casting.

- Since there is potential data loss, narrowing conversion require explicit casting by the programmer.

i.e. Manual (Explicit) Conversion

double → float → long → int → short → byte

double → long → int → char

```
eg. long no = 100L;
    int no1 = (int) no;   // 100
```

~~&~~

5. Provide examples of narrowing & widening conversions between primitive data types.

Ans Widening Conversion (Automatic / Implicit)

```
1) int no = 100;
   long no1 = no;        // 100
2) float no = 9.78f;
   double no1 = no;      // 9.78
3) char ch = 'A';
   int no = ch;          // 65
```

Narrowing Conversion (Manual / Explicit)

```
1) long no = 1000L;
   int no = (int) no;    // 1000
2) double no = 9.87654321;
   float no = (float) no;   // 9.876543
3) double no = 9.78;
   int no1 = (int) no;   // 9
```

6. How does Java handle potential loss of precision during narrowing conversions ?

Ans In Java, narrowing conversions involves converting a larger data type to a smaller data type. These conversion can lead to loss of precision because the target type may not have enough range or ~~pe~~ precision to represent the original value Java handles potential loss of precision in the following ways :

1) Explicit Casting is Required

Java does not allow implict narrowing conversions. To perform narrowing conversion, the programmer must use explicit casting. This ensure that the programmer is aware of potential risks of precision loss.

2) Truncation of Fractional Values

When converting from a floating-point type to an integer type, the fractional part is discarded. This results in truncation rather than rounding.

3) Overflow & Underflow

When ~~are~~ narrowing a larger integer type to a smaller integer type, if the value exceeds the range of target type, overflow or underflow occurs. The value wraps around using modulo arithmatic.

4) Loss of Precision in Larger Values (Floating → Integer)

5) Loss of Precision in Integer-to-Floating-Point Conversion (Long → Float)

7. Explain the concept of automatic widening conversion in java.

Ans In Java, an automatic widening conversion (also know as implicit type conversion) occurs when a smaller or less primitive data type is automatically converted into a larger or more precise type. This conversion happen without need of explicit casting by programmer & ~~is~~ it considered safe because no loss of data or precision.

Key points :

1) Implicit Conversion
2) No risk of data loss.
3) From smaller to larger types

byte → short → int → long → float → double
char → int → long → float → double

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

Ans Widenining & narrowing conversions can affect type compatibility & data loss in following ways :

1) Widening Conversion

These conversion are safe & preserves the source value, but can change it's ~~re~~ representation.

eg. Converting from integer type to decimal or from char to String.

PAGE No.
DATE / /

2) Narrowing Conversion

These conversion can loss data & require explicit conversion from larger to smaller types. For eg. Converting a fractional value to a integer type rounds the value Or converting numeric type to Boolean reduces it to either True or false.

3) Type Compatibility Implications

- Widening conversion are type-compatible, meaning java allows smaller types to be automatically promoted to ~~longer~~ larger types.
- Narrowing conversion are not-type-compatible by default & require explicit casting because not all values of a larger type can fit into a smaller type.

4) Data Loss Implications

- Widening conversion do not result data loss.
- Narrowing conversion can cause data loss in several ways: Truncation
  Overflow

5) Performance Implications

- Widening Conversion, generally have no significant performance cost because they are handled efficiently by JVM.
- Narrowing Conversion may incur a small performance cost due to the need for explicit casting & potential checks for overflow or truncation.