2408405-20001 Shweta Ma'am

Assignment No: 04

Q.1) What does the static keyword mean in Java?

⇒ Ex 1) In Java, static keyword is used to define members [variables, methods or nested classes] that belong to the class itself, rather than to any specific instance of the class.

Q.2) Difference between Static & Non Static Methods

Static Methods	Non Static Methods
1) Belong to class: → These methods are associated with class itself, not with any particular instance of class	1) Belong to an object: → These methods are associated with an instance of the class and they can access & modify the instance variable of that object.
2) Accessed using classname → We can call static methods directly on class w/o creating an obj.	2) Accessed through an object. → We need to create an obj of the class to call o/ non-static methods/variables of class.
3) Cannot access instance variable → Static methods can only access static variables & other static members of class. [They cannot access instance variables or methods directly, as they don't have access to any specific instance.]	3) Can access static variables → Non static methods can access both instance variable & static variables of class

behaviour

ii) Used for utility methods → Static methods are often used for utility functions, such as mathematical calculations/helper methods that don't depend on the state of any object.

ii) Used for object specific → Non static methods are used to define behaviours that are specific to individual object to the class

Q.2) What is the role of the static keywords in the context of memory management? (Geeksforgeeks.in)

⇒ i) Static keyword ensures that a variable or methods is allocated memory only once per class. meaning (it is shared across all instance of that class) (rather than being created separately for each object), thus optimizing memory usage by avoiding redundant allocations.

2) keypoints:

A] Single copy:

- When a variable is declared as 'static', only one copy of the variable exists for the entire class, regardless of how many objects are created from the class.

B] Class level access

- We can access a static variable directly using the class name, without needing an obj instance.

C] lifetime of the program:

- Static variables are allocated memory when the program starts, remain in memory until the program ends.

Overloading ⇒ compile time polymorphism
Overriding ⇒ Run time polymorphism

[Yes]

Q.5) Can static method be overloaded & overridden in Java? How static variables shared across multiple instances of a class?

→ A] Overloading:
1) Overloading is also a feature of OOP langs like Java that is related to compile time (or static) polymorphism.
2) This feature allows different methods to have the same name but different signatures, especially the no. of i/p parameters & type of i/p parameters.

B] Overriding:
1) Overriding is a feature of OOP languages like Java, related to run time polymorphism
2) A subclass provides a specific implementation of a method in the superclass (base class) (derived class)
3) The implementation to be executed is decided at run time & a decision is made according to the object used for call.

C] @ Part 2:
1) Static variables are shared across multiple instances of a class because they are initialized once the class is loaded and retain their values across all instances.
2) They can be accessed using the class name or directly if accessed from within the same class.

Q.4) What is the significance of final keyword in Java?

⇒ a) final variables [useful for creating constants].

- It means that the var value cannot be changed once it has been initialized.

b) Final Methods

- When applied to method, it prevents method from being overridden in subclasses.
- This is useful for ensuring that a particular behaviour is not altered in derived classes.

c) Final classes:

- It prevents the class from being inherited.
- This useful for creating classes that cannot be extended, typically for security or design reasons.

- Benefits:

a) Immutability:

→ It helps create ⇒ Immutable Object → characteristics
- Inherently thread safe
- easier to reason about

b) Security:

→ It prevents unintended modifications

c) Performance:

→ The compiler can sometimes optimize code that uses "final" entities, potentially leading to faster execution.

Q.5) What are narrowing & widening conversions in Java?

→ A] Narrowing Explicit Conversion (Explicit)

→ When you convert a large data types to a smaller data type

→ We need to explicitly cast the value of to the smaller type, as there is risk of data loss.

→ If the value of is too large to fit in the smaller type, it is truncated or produce errors.

Ex:-

```
double a = 10.5;
int b = (int) z;
SOP(a); ⇒ result ⇒ a = 10
```

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

B] Widening Conversion (Implicit)

→ When you convert a smaller data type to a large data type.

→ Java handles this automatically, so not explicit casting is required

→ There is no loss of information during this conversion.

Ex:-

```
int x = 10;
double y = x;
SOP(y); O/P = 10.0
```

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

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

→ i) Narrowing conversion occurs when you convert a larger data type to a smaller one. [int → short]
ii) These conversions can lead to potential loss of precision or data. [So Java requires explicit casting to indicate that you understand & accept the risk]
iii) Java handles it by providing with Compiler Warnings. [A warning when a narrowing conversion might lead to loss of precision, but if explicit casting, the code will compile.

Q.7) Explain the concept of automatic widening conversion in Java.

→ 1) An automatic widening conversion happens when a value of smaller data type is automatically converted to a larger data type w/o the need for explicit casting.
2) It is safe because the larger data type can accommodate all possible values of the smaller data type w/o any risk of data loss.
3) Automatic widening conversion in Java ensures that smaller data types can be safely & implicitly converted to larger data types, eliminating the need for explicit casting & ensuring no loss of data or precision.

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

→ A] Widening conversions: [Implications]

→ 1] Type compatibility:

• Widening conversions are generally safe & automatic because a smaller type can always fit into a large type w/o any risk of losing data.

• For ex:-

```
int a = 100;
long b = a ; // int to long
```

→ 2] No Data loss:

• Since the larger data type can hold all the values of the smaller type, there is not risk of data loss.

B] Narrowing Conversions:

→ 1] Type compatibility

• Narrowing conversions are not automatic & require explicit casting because there is a risk that the larger type might not fit into the smaller type.

• It can lead to compatibility issues if not handled properly.

=For ex:

```
long a = 1000L;
int b = (int) a ; // Explicit cast needed for
                     narrowing conversion.
```

• Here an explicit cast is necessary because long is not directly compatible with int.

2] Potential Data loss:
- Narrowing conversions can result in data loss if the value in the larger type exceeds the range of smaller type.

for ex:-
```
int a = 130;
byte b = (byte) a; // Data loss; b will hold -126.
```

3] Precision Loss:
- When narrowing from a floating point type to an integer, the fractional part is discarded, leading to a loss of precision.

For ex:
```
double a = 9.99;
int    b = (int) a; result is 9
```