ASSIGNMENT 6

Page No.
Date

1) What is method overloading in Java & explain with an example

- Method overloading in Java is a feature that allows a class to have more than one method having the same name, if their argument lists are different, for example void demo (int a) and double demo (double a). It's a way to increase the readability of the program.

eg.

```
import java.util.Scanner;
void demo (int a)
{
   System.out.println("a: " +a);
}
double demo (double a)
{
   system.out.println("a: " +a);
   return a*a;
}
demo(10);
demo(10.10);
```

output:

a: 10
a: 10.1

In this example, we have two methods named demo. The first demo method takes an integer as a parameter, while the second demo method takes a double.
When we call demo(10), the first method is invoked, and when we call demo(10.10), the second method is invoked. This is the essence of method overloading in Java.

2) What are the rules for method overloading resolution in Java? How does Java determine which 'overloaded method to call?

——
- The overloaded & overloading methods must be in the same class.
- The method parameters must change: either the number or the type of parameters must be different in the two methods.
- The return type can be freely modified
- The access modifier (public, private) can be freely modified

- Throw exceptions, if any, can be freely modified.

Java determines which overloaded method to call based on the arguments passed to the method during the invocation at compile time by matching the method signature, with the arguments provided. It selects the most specific applicable version of the overloaded method based on the types of the arguments. If no exact match is found, it tries to find the closest compatible match, otherwise we receive error.

3) What does the static keyword mean in Java? Explain the difference between static & non static methods

— The static keyword in Java is used to declare a member that belongs to the class itself, rather than to instances of the class. This means that the static member is shared among all instances of the class.

Static	Non-static
Static methods are associated with the class itself and are loaded into memory when the class is loaded.	Non-static methods are associated with instances of the class and are loaded into memory when an obj of class is created.
– Static methods can be invoked directly using the class name, without the need to create an instance of the class.	– Non-static methods are invoked on instances of class, meaning you need to create an obj. of class to call these.
– They can only directly access other static members of class.	Non-static methods can only directly access both members of class, including instance variables & other non-static methods.

4) Can static methods be overloaded & overridden in Java? How are static variables shared across multiple instances of a class?

— yes, static methods in Java can be overloaded but can not be overridden.
- Static method can be overloaded, meaning you can define multiple static methods within the same class with the same name but different parameter lists.
- However, static methods cannot be overridden in Java.

- when a class is loaded into memory, static variables are allocated memory space in a special area known as "method area". This memory space is shared by all instances of the class.

The step-by-step flow:

Class loading
↓
Memory Allocation
↓
Accessing static variables.
↓
Shared Access
↓
Lifetime

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

— The 'static' keyword in Java plays a significant role in memory management by influencing how memory is allocated and accessed for certain elements within a program.

- Allocating memory space for static variables and methods in the method area where the class is loaded.
- Allowing static variables to be shared among all instances of a class, contributing to the overall memory footprint.
- Enabling direct access to static methods without the need for object instantiation, reducing memory usage associated with creating instances.
- Executing static blocks during class loading for initializing static variables or performing one-time setup tasks.

6) what is the significance of the final keyword in Java?

→ The 'final' keyword in Java signifies immutability and prevents modifications of variables, methods or classes, enhancing code clarity, performance, and security.

- keyword 'final' when applied to a variable, it means the variable can only be assigned a value once. once initialized, its value cannot be changed.
- when applied to a method, it means the method cannot be overridden by subclasses.
- when applied to a class, it means the class cannot be subclassed, i.e cannot be extended by other classes.

7) Can a final method be overridden in a subclass? How does the final keyword affect variables, methods and classes in Java?

→ No, a final method cannot be overridden in a subclass in Java. The 'final' keyword

applied to a method declaration indicates that the method is not intended to be overridden by subclasses. Attempting to override a final method will return error (compilation error).

1) variable: It means the variable can only be assigned a value once after using final keyword. Once initialized, its value cannot be changed.

2) Method: When applied to a method, it means the method cannot be overridden by subclass.

3) class: When applied to a class it means the class cannot be subclassed. It cannot be extended by other classes.

8) What does the this keyword represent in Java? How is this keyword used in constructors and methods?

— The 'this' keyword in Java represents a reference of the

current instance of a class. It is commonly used within instance methods and constructors to refer to the current object on which the method is invoked or within which the constructor is being executed. This keyword is particularly useful in scenarios where there is a need to disambiguate between instance variables and methods parameters or to invoke one constructor from another constructor within the same class.

1) Constructors:
- When used in a constructor, "this" refers to the current object being initialized by that constructor.

eg.

```
public class xyz {
    private int x;

    public xyz (int x) {
        this.x = x;
    }
}
```

e) Methods: used to refer to the current obj on which the method is invoked.

eg.

```
public class Example
{
    private int x;
    public Example(int x)
    {
        this.x = x;
    }
    public void printX()
    {
        System.out.println(this.x);
    }
}
```

9) what are narrowing and widening conversions ~~between primitive data types~~ in Java?

—

1) Narrowing conversions:
- Also known as explicit conversion or casting, narrowing convensions occurs when you convert a data type with a

larger range to a data type with a smaller range.
- This conversion may result in loss of data or precision, so it requires an explicit cast operator in Java.
- eg.
long big = 1000L;
int small = (int) big.

2) Widening conversion:
- also known as implicit conversion widening conversion occurs when you convert a data type with a smaller range to a data type with a larger range.
- This conversion is done automatically by the Java compiler because it does not results in loss of data.
eg. byte small = 10;
int large = small;

10) Provide examples of narrowing and widening conversions in Java.

— (1) Narrowing conversion:
- Explicitly casting from a larger integer type to a smaller int type:

```
long big = 1000L;
int small = (int) big;
```

- Explicitly casting from a floating-point type to an integer type.

```
double floatingPoint = 3.14;
int integer = (int) floatingPoint;
```

(2) Widening conversion examples:

- converting from a smaller integer type to a larger int type:

```
byte small = 10;
int large = small;
```

- converting from an integer type to a floating-point type.

```
int integer = 42;
double floatingPoint = integer;
```

(11) How does Java handle potential loss of precision during narrowing conversions?

— In Java, when narrowing conversions may results in a loss of precision (eg. when converting from a longer data type to a smaller one like from double to int), the language truncates the higher-order bits of the value. This truncation means that only the lower-order bits that fit within the smaller data type are retained. However, Java does not perform any rounding during narrowing conversions, so the result may not accurately represent the original value.

eg.

```
int intValue = 123456789;
float floatValue = intValue;

double doubleValue = 3.14;
int intValue = (int) doubleValue;
```

12) Explain the concept of automatic widening conversion in Java.

— Automatic widening conversions in Java occurs when a value of a smaller data type is assigned to a variable of a

larger data type. This conversion is automatic because it does not require any explicit casting and is handeled by the Java compiler.

The key idea is that Java allows you to assign a value of a smaller data type to a variable of a larger data type without any extra effort from the programmer. This is possible because the larger data type can represent a wider range of values than the smaller data types, so no risk of losing information.

13) What are the implications of narrowing and widening conversions on type compatibility and data loss?

→ Widening conversions in Java involve promoting values to larger data types without losing data. This ensures compatibility and

safety since larger data types can accommodate a wider range of values. For example, assigning an integer value to a double variable automatically widens the integer to fit into the double without data loss.

On the other hand, narrowing conversions may results in data loss when values are converted to smaller data type. This occurs when the original value exceeds the range of the target data type. For instance, converting a double to an int truncates the decimal part, potentially losing precision.

In summary, widening conversions are safe and preserve data integrity, while narrowing conversions require attention to avoid data loss & maintain precision.