CDAC Mumbai PG-DAC MARCH 24

Assignment No- 6

1) What is method overloading in Java & explain with an example.?
Method overloading in Java refers to the ability to define multiple methods in a class with the same name but with different parameter lists. These methods can have different numbers, types, or order of parameters. Java determines which method to call based on the arguments passed during the method invocation.

```
    public int add(int a, int b) {
        return a + b;
    }

    // Method to add three integers
    public int add(int a, int b, int c) {
        return a + b + c;
    }
```

2) What are the rules for method overloading resolution in Java? How does Java determine which overloaded method to call?
Java distinguishes overloaded methods based on the number of parameters they have. If two methods have a different number of parameters, they are considered different overloads.: If two methods have the same number of parameters, Java considers the data types of those parameters. It looks for the closest match in terms of data types. Java will try to use the most specific method that matches the provided argument types. If an exact match is not found, Java will try to use the method that requires the least implicit type conversion.Java also considers the order of parameters when determining which overloaded method to call. If two methods have the same number of parameters and the same data types, but in a different order, Java can differentiate between them.If an exact match is not found, Java can perform type promotion to match an argument with a parameter. For example, if a method expects a float parameter but is called with an int, Java will promote the int to a float to match the method signature.Java ensures that the chosen method is compatible with the types of the arguments being passed. It does not allow ambiguous situations where multiple overloaded methods could potentially accept the same arguments.Overloaded methods can have different return types, but Java does not consider the return type when resolving which overloaded method to call. The method signature, which includes the method name and parameter types, is the primary determinant.By following these rules, Java determines which overloaded method to call at compile time, ensuring that the invocation is unambiguous and resolves to the most appropriate method based on the provided arguments.

3) What does the static keyword mean in Java? Explain the difference between static and non-static methods.
In Java, the static keyword is used to declare members (methods, variables, blocks) of a class that belong to the class itself, rather than to instances of the class. When a member is declared as static, it means there is exactly one instance of that member shared by all instances of the class. The static keyword can be applied to variables, methods, nested classes, and initialization blocks.

Static methods are associated with the class itself rather than with any specific instance of the class. They can be invoked directly using the class name, without the need to create an instance of the class.
Static methods cannot directly access instance variables or methods because they do not belong to any instance. They can only access static variables or other static methods.
Static methods are typically used for utility functions or operations that do not depend on the state of any particular instance.
Static methods are associated with the class and are loaded into memory when the class is loaded. They are shared among all instances of the class.

Non-static methods belong to individual instances of the class. Each instance has its own copy of instance variables and methods.
Non-static methods can access instance variables and other instance methods directly.
They must be invoked on a specific instance of the class using that instance’s reference.
Non-static methods can also access static members of the class.
Non-static methods belong to individual instances of the class and are loaded into memory when an instance of the class is created. Each instance has its own copy of non-static methods.

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

Yes, static methods can be overloaded and overridden in Java, but their behavior differs from instance methods:

Static methods can be overloaded just like instance methods. This means you can define multiple static methods in a class with the same name but different parameter lists.
Java determines which overloaded static method to call based on the method signature, similar to instance methods.

Static methods cannot be overridden in the same way instance methods are overridden.
When you declare a static method in a subclass with the same signature as a static method in the superclass, it’s not considered overriding; instead, it’s method hiding.
Method hiding means that the subclass defines a new static method with the same signature as the superclass method, effectively hiding the superclass method rather than overriding it.
The method to call is determined at compile-time based on the reference type.
Regarding how static variables are shared across multiple instances of a class:

Static variables (also known as class variables) are associated with the class rather than with instances of the class.
When you declare a static variable in a class, there is only one copy of that variable shared by all instances of the class.
Static variables are initialized only once when the class is loaded into memory, and they retain their value across multiple instances of the class.
All instances of the class access the same static variable, and any modifications made to the static variable are visible to all instances of the class.
Static variables are typically used for values or properties that are shared by all instances of the class, such as constants or counters.

5) What is the role of the static keyword in the context of memory management.
Static variables are allocated memory when the class is loaded into memory by the Java Virtual Machine (JVM).
They are initialized only once, regardless of the number of instances created or the execution flow of the program.
Static variables are shared among all instances of the class.
They exist independently of any instance of the class and can be accessed using the class name itself.
Static variables have a lifetime equivalent to the lifetime of the class itself. They remain in memory as long as the class remains loaded.
They are typically initialized when the class is loaded and remain in memory until the class is unloaded.
Static variables can be accessed directly using the class name, even without creating an instance of the class.
They are accessible from both static and instance methods of the class.
tatic variables are eligible for garbage collection only when the class loader responsible for loading the class is garbage collected. This typically happens when the class loader itself becomes unreachable.

Static variables can potentially extend the lifetime of objects they reference, preventing those objects from being garbage collected if they are not properly managed.

6) What is the significance of the final keyword in Java? 7)Can a final method be overridden in a subclass?How does the final keyword affect variables, methods, and classes in Java?
In Java, the final keyword is used to apply restrictions on classes, methods, and variables.

When applied to a variable, the final keyword indicates that the variable’s value cannot be changed once assigned. It essentially makes the variable a constant.
If the variable is a primitive type, its value cannot be altered after initialization.
If the variable is a reference type (e.g., an object reference), the reference cannot be changed after initialization. However, the state of the object it refers to can be modified.
Final variables must be initialized either at the time of declaration or within the constructor of the class.

When applied to a method, the final keyword indicates that the method cannot be overridden by subclasses.
Final methods provide a way to enforce immutability in the method’s behavior, preventing subclasses from modifying or extending the method’s functionality.

When applied to a class, the final keyword indicates that the class cannot be subclassed. It becomes a final class, and no other class can extend it.
Final classes are often used for utility classes or classes whose functionality should not be extended or modified.

8) What does the this keyword represent in Java?How is the this keyword used in constructors and methods?
In Java, the this keyword is a reference to the current instance of the class. It is used to refer to instance variables, instance methods, and constructors within the same class.

When there is a local variable with the same name as an instance variable, using this allows you to refer to the instance variable explicitly, distinguishing it from the local variable.
This is particularly useful in constructors and setter methods where parameters have the same name as instance variables.
In method chaining or fluent interfaces, this is used to return the current instance of the class from a method. This allows method calls to be chained together.

9) What are narrowing and widening conversions in Java?
Widening conversion, also known as implicit or automatic conversion, occurs when data of a narrower type is converted to a wider type.
It is safe and lossless because the target type can represent all possible values of the source type without losing information.
Java performs widening conversions automatically, without requiring explicit casting.
Examples of widening conversions include:
Converting from a smaller numeric type to a larger numeric type (e.g., int to long, float to double).
Assigning a subclass instance to a superclass reference (e.g., assigning a Child object to a Parent reference).

Narrowing conversion, also known as explicit or manual conversion, occurs when data of a wider type is converted to a narrower type.
It may result in loss of precision or data truncation because the target type may not be able to represent

all possible values of the source type.
Narrowing conversions require explicit casting in Java to indicate that the programmer is aware of the potential loss of data.
Examples of narrowing conversions include:
Converting from a larger numeric type to a smaller numeric type (e.g., double to int, long to short).
Assigning a superclass instance to a subclass reference, which requires explicit casting and may result in a runtime error if the object is not actually an instance of the subclass.

10) Provide examples of narrowing and widening conversions between primitive data types.

```
// Widening Conversion
int x = 10;
long y = x; // Implicit widening conversion from int to long

// Narrowing Conversion (Requires Explicit Casting)
double a = 10.5;
int b = (int) a; // Explicit narrowing conversion from double to int

// Narrowing Conversion with Potential Data Loss
double c = 1000.987;
int d = (int) c; // Explicit narrowing conversion from double to int, data loss (d = 1000)

// Subclass to Superclass Conversion (Widening)
Parent parent = new Child(); // Widening conversion from Child to Parent
```

11) How does Java handle potential loss of precision during narrowing conversions?
In Java, when performing narrowing conversions that may result in loss of precision (such as converting from a floating-point type to an integer type), the language provides rules for handling this situation.
Java truncates the fractional part of the floating-point number during narrowing conversions.
This means that only the integer part of the floating-point number is retained, and the decimal part is discarded.
For example, when converting from double to int, Java simply removes the fractional part of the double value.
If the fractional part being discarded is greater than or equal to 0.5, Java rounds the result towards negative infinity (i.e., towards the smaller integer).
If the fractional part is less than 0.5, Java rounds the result towards zero.
This behavior ensures that the conversion process is predictable and consistent.
Java allows narrowing conversions to be performed explicitly using casting, even though they may result in loss of precision.
The responsibility for handling potential data loss lies with the programmer, and Java does not raise a compilation error when narrowing conversions are explicitly performed.

```
EX. double doubleValue = 1234.5678;
 int intValue = (int) doubleValue;
```

In this example, when doubleValue is explicitly cast to an int, the fractional part (.5678) is discarded, resulting in the integer value 1234. Java performs the narrowing conversion without raising a compilation error, but the loss of precision is evident. It’s important for developers to be aware of potential data loss when performing such conversions and ensure that their code behaves as expected.

12) Explain the concept of automatic widening conversion in Java.
Automatic widening conversion, also known as implicit widening conversion, is a type conversion mechanism in Java where data of a narrower type is automatically converted to a wider type without requiring any explicit casting by the programmer. This conversion is considered safe because the target type can represent all possible values of the source type without loss of precision or data.
Automatic widening conversion applies primarily to numeric types, including integral types (byte, short, int,

long) and floating-point types (float, double).
When a value of a narrower numeric type is assigned to a variable of a wider numeric type, Java automatically performs the conversion.
The source type (the type of the value being assigned) is narrower than the target type (the type of the variable being assigned to).
For example, assigning an int value to a long variable or assigning a float value to a double variable.
Unlike narrowing conversions, which require explicit casting, automatic widening conversion does not require any explicit casting syntax in Java.
Java performs automatic widening conversion implicitly, allowing for cleaner and more concise code.

Ex.
Assigning an int value to a long variable.
Assigning a float value to a double variable.
Passing an int argument to a method parameter of type long.
Returning a float value from a method with a return type of double.

13) What are the implications of narrowing and widening conversions on type compatibility and data loss?
Widening conversions involve converting data from a narrower type to a wider type.
Since the target type (wider type) can represent all possible values of the source type (narrower type) without loss of precision, widening conversions are always safe and compatible.
For example, assigning an int value to a long variable or passing an int argument to a method parameter of type long are both widening conversions and are compatible operations.
Widening conversions do not result in data loss because the target type (wider type) can represent all possible values of the source type (narrower type) without loss of precision.
The conversion simply involves extending the range or precision of the data without any loss of information.
For example, converting from int to long or from float to double using widening conversion does not cause any data loss.

Narrowing conversions, on the other hand, involve converting data from a wider type to a narrower type.
The target type (narrower type) may not be able to represent all possible values of the source type (wider type) without loss of precision, which can lead to potential data loss.
As a result, narrowing conversions are potentially unsafe and may cause compatibility issues if not handled carefully.
For example, assigning a double value to an int variable or passing a double argument to a method parameter of type int are narrowing conversions and may lead to data loss or runtime errors.
Narrowing conversions may result in data loss because the target type (narrower type) may not be able to represent all possible values of the source type (wider type) without loss of precision.
When a wider type is converted to a narrower type, the fractional part of floating-point numbers may be truncated, or the value may be rounded, leading to potential loss of information.
For example, converting from double to int or from long to short using narrowing conversion may result in data loss if the fractional part of the value is discarded or if the value is rounded.