# Assignment 6

1) What is method overloading in Java with an example?

→ Method overloading in Java refers to the ability to define multiple methods in a class with same name but different parameter. This allows you to have methods with same name performing different task based on the input parameter. Method overloading is determined by the number, type & order of parameters in the method signature.

example

```
public class Calculator{

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

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

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

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

```
public static void main (String [] args) {
    Calculator calculator = new Calculator ();
System.out.println ("Sum of 5 and 10 = " + calculator.add(5,10));

System.out.println ("Sum of 5, 10 and 15 = " + calculator.add(5,10,15));

System.out.println ("Sum of 3.5 and 2.5 = " + calculator.add(3.5, 2.5));

System.out.println ("Sum of "Hello" and "World" = " + calculator.add
                                                    ("Hello", "World"));
}
}
```

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

→

Method overloading resolution is determined by the compiler based on the method signature. When you call an overloaded method, Java examines the method signature to determine which version of the method to execute.

Here are the rules for method overloading

(1) Number of Parameters: Java distinguished overloaded methods based on the Number of parameters. If two methods have the same name but a different number of parameter, Java can determine which method to

call based on the number of arguments provided in method call.

(2) Data Type of parameter: If Two or more methods have same name and same number of parameters, Java distinguishes them based on the data types of parameter

(3) Type promotion: If there is no exact match for the parameter type, Java will try to find the closest match by applying implicit type conversions known as type promotion. Type promotion includes widening conversion & autoboxing

(4) Varags & Array types: If a method is overloaded with a varags parameter & there is another overloaded method with the same name and the same number of parameters but of a different type, Java will given preference to the non varargs method

(5) Return type: Method overloading resolution is not based on the return type. Two methods with same name & parameter but different return type are not considered overloaded & will result in compilation error.

3) What does the static keyword mean in Java? How the difference between static & non-static method.?

In Java, the static keyword is used to declare members (variables and methods) that belongs to the class itself than to instance of the class. Here's what it mean in different contexts:

1. Static variables
- When a variables is declared as static within a class, it becomes a class variable. This means that the variable is associated with the class itself, rather than with instances of the class. There will be only one copy of a static variable shared among all instances of the class.
- Static variables are initialized only once, at the start of program's execution. & retain their values throughout the program's lifetime.
- Static variables are accessed using the class name, followed by the dot operator, without needing to create an instance of the class.

2. Static Methods:
- When methods are declared as 'static' within a class, it becomes a class Method. This means that the method is associated with the class itself, rather than with instances of the class. Static method can be called directly using class name, without the need to create instance.
- Static method cannot access non static (instance) variables directly nor can they call non static method without first creating an instance of the class.

- Static methods are commonly used for utility methods which perform generic operations and do not require access to instance-specific state.

| Static | Non static |
| --- | --- |
| Static methods are invoked using the class name, such as 'ClassName.methodName()'. | Non-static methods are invoked using the class name of the class instance of the class such as 'instanceName.methodName();' |
| Static method can only directly access other static members (variables & methods) of the class | Non static methods can directly access both static & non static members of class. |
| Static method and variables are stored in a common memory are shared by all instances of the class | Non static methods and variables are stored in memory allocated seperately for each instance of the class |
| Static method are often used for operations that don't required access to instance-specific data or behaviour | Non-static methods are used for operations that involve instance-specific data or behaviour |

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

→ yes, static method can be overloaded, but not overridden in Java.

## Overloading Static Methods:

## Overridding static Method.

- Overriding: In Java, method overriding occurs when a subclass provides a specific implementation of a method that is already provided by its superclass. However, static methods cannot be overridden in Java, they only be hidden.

- When you define a static method in a subclass with the same signature as a static method in the superclass, it hides the superclass's static method but does not override it.

## Sharing Static Variables Across Multiple Instances:

- Static Variables: Static variables (also known as class variables) are variables that belong to the class rather than any instance of the class. They are shared among all instances of the class.

- Static variables are initialized only once, at the start of the execution, and they retain their value throughout the program's execution.
- Any change made to a static variable will reflect in all instances of the class because they share the same memory location.

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

→ The 'static' keyword plays a significant role in determining how memory is allocated and managed for variables & methods.

1. Class level Allocation :
When variable or method is declared as 'static', it becomes associated with class. which means only one copy of the static variable or method exists in memory, regardless of how many instance of class is created.

2. Memory Effecency : Since static members are associated with class itself, they are stored in method area. The area is shared among all instance of class. Thus, using static member can be more efficient, especially when the same data or behaviour is shared across multiple instance of class.

③ Lifetime - Variable & methods have a lifetime that extends for the entire duration of program's execution.

④ Accessing from Anywhere: Static members can be accessed directly using class name, without needing to create instance of the class.

⑤ Thread Safety :- Since static variables are shared among all instance of a class, they can introduce thread safety concerns in multithreaded environments. Access to static variables should be synchronized if multiple thread may modify them concurrently to avoid data inconsistency issues.

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

→

① Final variables :-
- When final keyword indicates that variables value cannot be changed once it has been initialized.
- for primitive data type, the value of a final variables cannot be modified
- Using final variable can improve code, readability, maintainability & help prevent accidental changes to critical values or reference.

(2) Final Methods
'Final' keyword with method indicates that the method cannot be overridden by subclasses.

(3) Final classes.
This means class cannot be subclassed.

(4) Final parameter

The parameter's value cannot be modified with in the method.

Overall, the final keyword in Java provided a mechanism for enforcing immutability, ensuring method or class behaviour, & preventing subclassing or method overrides in specific contexts. It enhances code robustness, clarity, and security by imposing certain constraints on variables, methods, and classes

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

→

No, a final method cannot be overridden in a subclass in Java. When a method is declared as final in a superclass, it means that the method's implementation is fixed & cannot be changed by any subclass.

Significance of the 'final' keyword:

(1) Variables: When applied to variables, the 'final' keyword indicates that the variable's value cannot be changed after initialization.

(2) Methods: When applied to methods, the 'final' indicates that the method cannot be overridden by any subclass.

(3) Classes: When applied to class, the final keyword indicates that the class cannot be subclassed, meaning no other class can extend it.

Effect on Variables, Methods & classes

(1) Variables: final' variable can not have their values reassigned after initialization. They behave as constant.

(2) Methods: 'final' method cannot be overridden by subclasses their implementation remains fixed.

(3) Classes: final classes cannot be extended by other classes. They are considered as complete and cannot have subclasses.

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

→ The 'this' keyword is a reference to the current object. It can be used with in a class to refer to instance variable, instance method, or to invoke constructors.

(1) Reference to Instance Variables.

Inside an instance method or a constructor, 'this' refers to the current object instance variable. This is particularly useful when there's a need to disambiguate between instance variable & local variable or parameters with the same name.

(2) Reference to Instance Method.

Inside a method, 'this' can also be used to call another instance method of the same object.

(3) Invocation of Constructor :-

Inside a constructor, 'this' is used to call another constructor of the same class. This is known as constructor chaining.

(4) Passing the current Object as a parameter

this can also be used to pass the current

object as a parameter to another method.

9 What are narrowing & widening conversions in Java?

→ They are related to the conversion of data between different data type. It occurs implicitly or explicitly & are categorized based on the size & precision of the data types involved.

① Widening Conversion

- Also known as implicit conversion, widening conversion occurs when a value of a smaller data type is automatically promoted to a value of a large data type with out any loss of information.
- It ensure that widening are always safe because no data loss occurs when converting from a smaller to larger data type.

② Narrowing conversion

- Also known as explicit conversion or type casting, narrowing occurs when a value of larger is explicitly converted to a value of small data type, potentially resulting data loss.
- risk of losing data.

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

→ Widening Conversion

(1) From byte to short

```
byte a = 10;
short b = a;
```

(2) From short to int

```
short c = 100;
int d = c;
```

(3) From int to long

```
int e = 1000;
long f = e
```

(4) From float to double

```
float c1 = 3.84f;
double c2 = c1;
```

Narrowing Conversion :-

(1) From int to short

```
int a1 = 1000;
short b1 = (short) a1;
```

(2) from long to int

```
long c1 = 10000L;
int d1 = (int) c1;
```

(3) from double to float

```
double e1 = 3.14;
float f1 = (float) e1
```

(4) From int to byte

```
int y1 = 100;
byte y2 = (byte) y1
```

11) How does java handles potential loss of precision during narrowing conversion?

→ Java handles potential loss of percision during narrowing conversion by truncating the data beyond the range or precision of the traget data type without rounding. It requires explicit casting when performing compilation. It's the programmer's responsibility to ensure that the conversion is safe & appropriate from the given context.

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

→ Automic widening conversion in Java refers to the implicit conversion of a smaller data type to larger one with out the need for explicit casting. Java performs automatic widening conversion s safely because no of data loss occurs when converting from smaller to larger data type.

example

Converting an 'int' to a 'long' or "float" to a "double" involves automatic widening conversion. This process ensure that are promoted to larger data types without loss of precision or magnitude

13) What are the implication of narrowing and widening conversion on type compatibility & data loss?

→ Narrowing conversion may result in potential loss of data precision as values are truncated to fit