What is method overloading in Java & Explain with an example.?

1) Method overloading in Java.

Method overloading in Java allows a class to have multiple methods with the same name but with different parameters. Java distinguishes between these method based on their number, type, and order of parameters. Example.

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

2) Rules for method overloading resolution:

Java determines which overloaded method to call based on the most specific matching parameters. If there's an exact match, Java selects that method. If not, it looks for the next closest match based on parameter types.

3) Static keyword in Java.

→ In Java, the static keyword is used to declare members that belong to the class rather than to instance of the class. Static members are shared among all instances of the class. Non-static methods and variables belong to individual instance of the class.

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

4) Static method in Java:-

- Static methods can be overloaded, but not overridden. Overloading allows defining multiple methods with the same name but different parameter lists. Static variables are shared across all instances of a class. meaning changes to a static variable made by one instance affect all other instance.

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

→ The static keyword in Java ensures that only one Instance of a variable exists in memory regardless of how many instance of the class are created. It's allocated memory at compile-time and remains in memory throughout the execution of the program.

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

→ The final keyword in Java is used to restrict the user from changing the value of a variable overriding a method or inheriting from a class. It ensures that a variable can only be initialized once. a method cannot be overridden and class cannot be subclassed.

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

→ The this keyword in Java refers to the current instance of the class. It is primarily used within constructors and methods ~~of~~ to refer to the current object. In constructors, this is used to differentiate between instance variable and parameters with the same name. In methods, this is used to call other constructors or methods of the same class.

9) What are narrowing and widening conversions in Java?

→ Narrowing conversion occurs when a data type with a larger range is converted to a data type with a smaller range, potentially losing information. Widening conversion is the opposite, where a data type with a smaller range is converted to a data type with a larger range, without loss of information.

10) Provide example of narrowing and widening conversion between primitive data types.

→ Narrowing conversion : int to short or float to int
Widening conversion:- byte to int or int to long.

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

→ Java handles potential loss of precision during narrowing conversions by truncating that extra bits. without raising any errors or exceptions. It's the responsibility of the programmer to ensure that the loss of prelision is acceptable for the applications requirements.

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

→ Automatic widening conversions occurs when a value of a smaller data type is assigned to a variable of a larger data type. Java automatically promotes the smaller data type to prevent loss of information.

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

→ Narrowing and widening conversions affect type compatibility and potential data loss. Narrowing conversions may lead to loss precision. while widening conversions are generally safe but can potentially lead to unexpected behaviour if not handled properly. It's crucial to understand these implications when designing and implementing Java applications