* what is method overloading in Java & explain with example?

→ If a class has multiple methods having same name but different Parameters.

→ Two ways to overload methods:
1. By changing number of arguments
2. By changing data type

1. method overloading: changing no. of arguments.

→ Two methods, first add() method perform addition of two numbers and seand add() method perform addition of three numbers.

eg.
```
class Adder {
    static int add(int a, int b){return a+b;}
    static int add(int a, int b, int c) {return a+b+c;}
}
class TestOverloading1{
    public static void main(String[] args){
    System.out.println(Adder.add(11,12));
    System.out.println(Adder.add(11,12,13));
}
}
```

2. method overloading: changing data type of arguments

first add method recieves two integers arguments and second add method recieves two double arguments.

```
class Adder{
static int add(int a, int b, ){return a+b;}
static double add(double a, double b){return a+b;}
}
```

```
class TestOverloading2{
public static void main(String[]args){
System.out.println(Adder.add(11,11));
System.out.println(Adder.add(12.3,12.6));
}
}
```

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

1. Java compiler always tries to choose the most specific method available with ~~at~~least number of modifications to the arguments.
2. Next point is about Boxing/unboxing Java developers wanted the old code to work exactly the same as it used to work before boxing/unboxing.
3. with points 1 and 2 in mind the order in which preference is given is -

Widening (size wise) [minimum first] > Boxing/Unboxing > variable arguments.

eg. public s v execute(int data){syst s.o.p("int");}
p s v execute(short data){s.o.p.("short");}
p s v execute(object data){s.o.p.("object");}
p s v execute(String data){s.o.p.("string");}

```
p s v main(...){
byte byt=3;
execute(byt);
execute(10);
```

→ output

short
int
object
string

- First call was; byte byt = 3
  execute(byt); // 1st call
  - As per the rule most preffered was would be widening. There are two option here. widen a byte (8bit) to a short (16 bit) or an int (32 bit). widening with lowest size preffered (which is short) & hence we got short printed.

- second call execute(10);
  direct call, we have execute method with int parameter.

- third call, integer i=6; execute(6);
  since we do not have a method with integer as argument we will go for next best case that is polymorphic superclass object is superclass of all objects implicitly. so we get object printed

- fourth call, execute("Test");
  - execute method for string which is chosen here

Note:- Java compilation will fail if there are no matches or ambiguous matches for a function call.

* what does the Static keyword mean in java? Explain the difference between static & non-static method.
  → The static keyword is a non-access modifier used for methods and attributes. Static methods/ attributes can be accessed without creating an object of a class.
  - The static keyword is used for memory management mainly.
  - The static keyword belongs to the class than an instance of the class.

- Static variable can be used to refer to the common property of all objects (which are is not unique for each object), for example, the company name of employees, college name of students, etc.
- The static variable gets memory only once in the class area at the time of class loading.
- memory efficient.

Static	Non-Static
- Associated with class itself.	- specific to each instance of a class, tied to objects created from class
- gets memory once, at the time of class loading. They are shared among all the instances.	- memory allocated seperately for each instance of the class. Each object has it's own copy of non-static members
- accessed directly using classname accessible from anywhere within the program.	- accessed using object reference followed by the member name. They are specific to particular instance of the class.
- has a global scope.	= local scope
- Static members can only access other static members within the same class. They can not directly access non-static members	- non-static members can access both static and non-static members within the same class. They have direct access to all members
- Commonly used for utility methods, constants or variables that are not specific to individual instances.	- non-static members are used for instance specific behaviour, as they hold data specific to each object.

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

→ Static method can be overloaded but not overridden. They can have different parameters while having the same name in the same class or subclass. They cannot be overridden because they act on the class itself, not an object.

→ when we declare a static field static, exactly a single copy of that field is created and shared among all instances of that class.

5) what is the role of the static keyword in the context of memory management?

→ A static member belongs to the class and not the instance. It can be called without creating an instance of the class.

→ It helps in memory management as static variables are shared among all instances, reducing the amount of memory required.

→ static also persist for the lifetime of the program, which can lead to memory leaks if not managed carefully.

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

→ It is a non-access modifier, used to indicate that variable, method or class cannot be modified or extended.

→ Applicable only to a variable, a method or a class.

→ used to restrict a user in java.

1) can a final method be overridden in a subclass? How does the final keyword affect variables, methods, and classes in java?

→ No. The methods that are declared as final can not be overridden or hidden.

→ variable :-

If you make any variable as final, you cannot change the final value of final variable (it will be constant).

eg.. final variable speedlimit, we are going to change the value of this variable, but it can't be changed becz final variable once assigned a value can never be changed.

```
class Bike9 {
    final int speedlimit = 90;
    void run () {
        speedlimit = 400; }
    public static void main (String args [ ]) {
        Bike9 obj = new Bike9 ();
        obj.run ();
    }
}
```

→ compile Time error.

→ method :- final method cannot be overridden.

```
class Bike {
    final void run () { System.out.println ("running"); } }
class Honda extends Bike {
    void run () {
        System.out.println ("running safely with 100 kmph");
    }
    public static void main (String args [ ]) {
        Honda honda = new Honda ();
        Honda.run ();
```

→ compile time error

→ class : final class cannot be extended.

```
final class Bike {}
class Hondai extends Bike {
void run () { System.out.println ("running safely with 100 kmph");
public static void main (String args[]) {
  Hondai honda = new Hondai ();
  honda.run ();
}                                    → Compile time error.
}
```

8] what does this keyword represent in java ? How is the this keyword used in constructors and methods ?

→ within an instance method or a constructor, this is a reference to the current object — the object whose method or constructor is being called. To can refer to any member of the current object from within an instance method or a constructor by using this.

Using this with a ~~constructor~~. field.

the most common reason for using the this keyword is because a feild is shadowed by a method or constructor parameter.

for eg.

when we pass value as arguments, compiler gives this keyword and we can use these values from parameter of method & constructor using this keyword, which referes to that particular object.

What are the narrowing and widening conversions in Java ?

Type casting is when you assign a value of one primitive data type to another data type.

In java there are two types of casting:

1] widening Casting (automatically) -

Converting a smaller type to a larger type size

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

- It is done automatically when passing a smaller size type to a larger size type:

```
public class main {
    public static void main (String [] args) {
        int myInt = 9;
        double myDouble = myInt; // Automatic casting:
                                 // int to double.
        S.O.P (myInt);           → 9
        S.O.P. (myDouble);       → 9.0
    }
}
```

2] Narrowing Casting : must be done manually by placing the type in parentheses in front of the value.

eg.

```
public class main {
    public static void main (String [] args) {
        double myDouble = 9.78d;
        int myInt = (Int) myDouble; // manual casting: double to int
        System.out.println (myDouble); → 9.78
        System.out.println (myInt);    → 9
    }
}
```

10] provide examples of narrowing and widening conversion between primitive data types

```
public class WideningTypeCasting Example {
    public static void main (String [] args) {
    int x = 7;
    long y = x;
    float z = 7;
    System.o.p ("Before conversion, int value " + x); → 7
    S.o.p ("After conversion, long value " + y); → 7
    S.o.p ("After conversion, float value " + z); → 7.0
    }
}
```

Narrowing Typ casting

```
public class NarrowingTypeCasting Example {
    public static void main (String args[]) {
    double d = 166.66;
    long l = (long) d;
    int i = (int) l;
    S.o.p ("Before conversion:" + d)  → 166.66
    S.o.p ("After conversion into long:" + l); → 166
    S.o.p ("After conversion into int type:" + i); → 166
    }
}
```

13] ~~what are~~

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

→ In java narrowing occurs when you convert a data type with a larger range to one with a smaller range. eg. double into int. may result into loss of precision.

- Java handles potential loss of precision during narrowing conversions by truncating the higher order bits of the value being converted.

1. Implicit Conversions : Java allows certain implicit conversions between numeric types, such as converting an int to a double without loss of precision.

2. Explicit casting: when larger type explicitly cast into smaller type, telling compiler that you are aware of the potential loss & intentionally performing the conversion.

eg. double d = 10.5

int i = (int) d; // Explicit cast, Potential loss of precision in this case, if the fractional part of 'd' is truncated, and 'i' will be assigned the integer part of 'd', which is 10.

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

→ Widening Conversion take place when two data types are automatically converted. This happens when:

- The two data types are compatible.
- when we assign a value of a smaller data type to a bigger data type.

Byte → Short → int → Long → float → Double.

what are the implications of narrowing & widening conversions on type compatibility and data loss ?

widening conversions preserve the source value but can change it's representation. This occurs if you convert from an integral type to decimal, or from char to string.

narrowing conversion changes a value to a data type that might not be able to hold some of the possible values. a fractional value is rounded when it is converted to an integral type, and a numeric type being converted to Boolean is reduced to their either True or False.