Methods

Overloading Methods

```
// overloading: having two (or more) methods with same name
// but different parameter list

public void greet (int x) { System.out.println("Hello"); }
public void greet (double x) { System.out.println("Good Afternoon"); }
public void greet (int x, int y) { System.out.println("Good Day"); }

greet(5);
  => Hello
greet(3.14);
  => Good Afternoon
greet(7, -11);
  => Good Day
```

```
// in passing argument doesn't exactly match the parameter type
// Java will pick the most similar version of the method

public void greet (int x) { System.out.println("Hello"); }
public void greet (double x) { System.out.println("Good Afternoon"); }
public void greet (int x, int y) { System.out.println("Good Day"); }

short a = 2;
greet(a);
  => there is no greet(short x), so Java looks for larger primitive type
  => Hello
```

```
public void greet (Short a) { System.out.println("Hi"); }
public void greet (Integer a) { System.out.println("Hello"); }
public void greet (String str) { System.out.println("Good Afternoon"); }
public void greet (Object o) { System.out.println("Good Day"); }


greet(2.3);
  => wraps 2.3 in Double, which extends Object -> "Good Day"
greet((short)2);
  => wraps (short)2 to Short -> "Hi"
greet((byte)3);
  => wraps (byte)3 to Byte, which extends Object -> "Good Day"
greet("John Wayne");
  => "Good Afternoon"
```

```
// Java will also look for supertypes


public void greet (Number a) { System.out.println("Hi"); }

public void greet (CharSequence a) { System.out.println("Hello"); }

public void greet (Object o) { System.out.println("Good Day"); }


greet(3.14);

  => wrap 3.14 to Double, which implements Number -> "Hi"

greet("Luke");

  => String implements CharSequence -> "Hello"

greet(new int[]{1, 2, 3});

  => can't find anything similar -> "Good Day"
```

```
// you cannot overload array with varargs !


public int doSomething(int[] nums) { };

public int doSomething(int... nums) { };

  => DOES NOT COMPILE


doSomething(new int[]{1, 2, 3, 4, 5});

  => it could be both
```

Conclusion

The order Java uses for finding the right overloaded methods:

1. Exact match by type
2. Larger primitive type
3. Autoboxed type
4. Varargs