Methods

Autoboxing

Failing cases

```
long p = 8;
Long pw = 8L;
// incompatible types: int cannot be converted to java.lang.Long
Long pw2 = 8;  //does NOT compile
```

final

```
    void varFinal() {
        //legal
        final var x = 5;
    }
```

Effective final

The test for effectively final is if the final modifier can be added to the local variable and the code still compiles.

varargs

Two varargs parameters are not allowed in the same method.
A method may contain **at most** one varargs parameter, and it must appear **as the last argument** in the list.

```
//won't compile!
public void bass(String... values, int... nums) {}
//won't compile!
public void hello(String... values, String desc) {}
```

varargs as variable

While varargs is used like an array from within the method, it can only be used as a method parameter!

```
String... value;  //does not compile!
```

Passing data among methods

Java is a "pass-by-value" language. This means that a copy of the variable is made and the method receives that copy. Assignments made in the method do not affect the caller.

Access to protected method. IMPORTANT!!!

```
package a;
class A {
  protected void hello(){}
}
```

```
package b;
class B extends A {

  public static void main(String[] args) {
    B b1 = new B();
    b1.hello(); //fine this compile!

    A b2 = new B();
    //THIS DOES NOT COMPILE!
    //b2.hello();
  }
}
```

In the main method I am in the package **b**, and I am trying to access to a protected method defined in the class A (the type is A, not B) which is in the package **a**, then a different package. Then it does not compile!

String to number

parseLong

```
var result = Long.parseLong("17")
//result is a primitive long
//System.out.println(result.toString()); //does not compile
```

valueOf

```
var result = Long.valueOf("17")
//result is a Long
System.out.println(result.toString()); //does compile!
```

Overloading

Overloading primitive types

Since there is no exact match, Java attempts to promote the primitive type to **double** before trying to wrap it as a Float.

```
void print(double d);
void print(Float f);

//caller
print(2F); //this calls the method print(double d)
```

Overloading primitives