

### The main() method

The main() method is the main entry point which Java invokes when you start a Java program from the command line. The command

```
java Distance 0 3 4 0
```

instructs Java to locate the Distance class, put the four command line arguments into an array of String values, then pass those arguments the public static main(String[]) method of the class. (We will introduce arrays shortly.) Any Java class that you want to invoke from the command line or desktop shortcut must have a main method with this signature.

```
public static void main(String[] args)
{
    Distance dist = new Distance(
        intValue(args[0]), intValue(args[1]),
        intValue(args[2]), intValue(args[3]));
    dist.printDistance();
}
```

The main() method invokes the final method, intValue(), four times. The intValue() takes a single string parameter and returns the integer value represented in the string. For example, intValue("3") will return the integer 3.

### The intValue() method

The intValue() method delegates its job to the Integer.parseInt() method. The main method could have called Integer.parseInt() directly; the intValue() method simply makes the main() method slightly more readable.

```
private static int intValue(String data)
{
    return Integer.parseInt(data);
}
```

This method is private since, like the fields point0 and point1, it is part of the internal implementation of the class and is not part of the external programming interface of the Distance class.

### Static vs. Instance Methods

Both the main() and intValue() methods are *static methods*. The static keyword tells the compiler to create a single memory space associated with the class. Each individual object instantiated has its own private state variables and methods but use the same **static** methods and members common to the single class object created by the compiler when the first class object is instantiated or created. This means that the method executes in a static or non-object context - there is no implicit separate instance available when the static methods run from various objects, and the special variable this is not available. As such, static methods cannot access instance methods or instance fields (such as printDistance()) or point0 directly. The main() method can only invoke the instance method printDistance() method via an instance reference such as dist.

## Data Types

Most declarations have a data type. Java has several categories of data types: reference types, primitive types, array types, and a special type, void.

### Reference Types

A *reference type* is a Java data type which is defined by a Java class or interface. Reference types derive this name because such values *refer to* an object or contain a *reference to* an object. The idea is similar to pointers in other languages like C.

Java represents sequences of character data, or *String*, with the reference type `java.lang.String` which is most commonly referred to as `String`. *String literals*, such as "Distance between " are constants whose type is `String`.

This program uses three separate reference types:

1. `java.lang.String` (or simply `String`)
2. `Distance`
3. `java.awt.Point`

For more information see chapter : Java Programming/Classes, Objects and Types.

### Primitive Types

In addition to object or reference types, Java supports *primitive types*. The primitive types are used to represent Boolean, character, and numeric values. This program uses only one primitive type explicitly, *int*, which represents 32 bit signed integer values. The program also implicitly uses *double*, which is the return type of the `distance()` method of `java.awt.Point`. *double* values are 64 bit IEEE floating point values. The `main()` method uses integer values 0, 1, 2, and 3 to access elements of the command line arguments. The `Distance()` constructor's four parameters also have the type `int`. Also, the `intValue()` method has a return type of `int`. This means a call to that method, such as `intValue(args[0])`, is an expression of type `int`. This helps explain why the `main` method cannot call

```
new Distance(args[0], args[1], args[2], args[3]) // this is an error
```

Since the type of the `args` array element is `String`, and our constructor's parameters must be `int`, such a call would result in an error because Java cannot automatically convert values of type `String` into `int` values.

Java's primitive types are *boolean*, *byte*, *char*, *short*, *int*, *long*, *float* and *double*, each of which are also Java language keywords.

### Array Types

Java supports *arrays*, which are aggregate types which have a fixed element type (which can be any Java type) and an integral size. This program uses only one array, `String[] args`. This indicates that `args` has an array type and that the element type is `String`. The Java VM constructs and initializes the array that is passed to the `main` method. See *arrays* for more details on how to create arrays and access their size.

The elements of arrays are accessed with integer indices. The first element of an array is always element 0. This program accesses the first four elements of the `args` array explicitly with the indices 0, 1, 2, and 3. (This program does *not* perform any input validation, such as verifying that the user passed at least four arguments to the program. We will fix that later.)

**void**

*void* is not a type in Java; it represents the absence of a type. Methods which do not return values are declared as void methods.

This class defines two void methods:

```
public static void main(String[] args) { ... }
public void printDistance() { ... }
```

**Beginner topics**

- Overview
- Getting started
- Language fundamentals
- Classes and objects
- Basic arithmetic
- Collections
- Exceptions
- Concurrent Programming
- Javadoc & Annotations
- Byte Code

**Appendices**

- Links
- Glossary
- Index

**User Interface topics**

- Basic I/O
- Event handling
- Intro to Swing
- Graphics
- Applets

**Extra Reading****Advanced topics**

- Networking
- Database programming
- Reflection
- Design patterns
- Libraries, extensions and frameworks
- 3D programming
- Threads

## Syntax

Navigate **Language Fundamentals** topic:

Java derives much of its *syntax* from the C programming language: basic assignment statement syntax, expressions, control flow statements and blocks, etc. will be very familiar to C programmers.

**Unicode**

Java source code are built by Unicode characters.

**Tokens**

Java programs consist of a sequence of different kinds of tokens. For example, there are word tokens such as **class** and **public** which are *keywords*.

**Keywords**

Those are special words with reserved meaning in Java. Those words can not be used by the programers to name identifiers.

**Identifiers**

Other words (non keywords) are identifiers. Identifiers have many different uses in Java but primarily they are used as names, class names, method names, and variable names... .

**literals**

Java also has tokens to represent numbers, such as 1 and 3; these are known as *literals*.

*String literals*, such as "http://en.wikibooks.org/Java\_Programming", consist of zero or more characters embedded in double quotes.