class Main {
    public static void main (String[] args) {
        System.out.println ("Hello world!");
    }
}
class Main {
    // String ... = String[] ; cute but rare
    public static void main (String... args) {
        System.out.println("Hello world!");
    }
}
public class Main {
    private Main() {} // disable instantiation
    public static void main (final String[] args) {
        System.out.println("Hello world!");
    }
}
// Class not intended to be used to create other classes

public final class Main {

    // All parameters should be final; enforced by 'checkstyle'
    public static void main (final String[] args) {
        System.out.println("Hello world!");
    }
}

Consider each word/token.

- What happens if you leave it out?
- What happens if you modify it?
• access modifier for things with unrestricted access; one public, top-level Java class per file
• modifier for classes that are not to be sub-classed
• keyword introducing a Java class
• name of class; capitalized by convention; should be same as file name
• access modifier for methods with unrestricted access; method `main` must be public if it is to be the starting point of the program by the Java virtual machine
• method modifier indication a non-instance method
• return type of void means the method does not return a value; it is a subprocedure not a function
• name of method; must be called "main" if it is to be the starting method (entry point)
• type of the one parameter to the method; must be an array of strings (or equivalently varargs), if the method is to be the starting method
• name of the one parameter to the method
• java.lang.System is the name of the class in package java.lang containing standard I/O objects
• Field of java.lang.System with type java.io.PrintWriter containing the object with the reference the program’s standard output stream.
• Name of the overloaded method that puts strings on to output stream (prints or displays the text on the screen).
Definitions.

- access modifier (e.g., public, private)
- entry points (starting point of execution)

Rules of thumb.

- Declare your (outer) classes public.
- One public class the same name as the file.
- Declare your classes final.
- Declare your formal arguments final.
- (No public constructors for utility classes.)
OS

Program

System.in
System.out
System.err

String[] args

System.getProperty
Long.parseLong

command line args

env
key value

standard IO package

JVM
OS Interface

- standard IO package; abstract and standardized
- command line arguments; simple
- environment map and also JVM properties; many OS dependent keys

- `System.in`, `System.out`, `System.err`
- `String[] args`
- `System.getenv()`, `System.getProperties()`
JVM has its own platform environment established in negotiation with the OS.

```
// The unmodifiable map of OS environment
Map<String, String> env = System.getenv();

// Less platform dependent are the Java system properties
System.getProperties().list(System.out);

// User can supply environmental values:
// * with JVM args ’$ java -Dseed=12345 Main’
// * with OS environment, eg, ’$ env seed=12345 java main’
String seed1 = System.getProperty("seed", "54321");
String seed2 = System.getenv().getOrDefault("seed","54321");
```
// A program can get values from Java system properties
System.out.println(System.getProperty("file.encoding"));

// Be careful arguing with the OS
System.setProperty("file.encoding", "Latin-1");

// System properties sometimes 'travel' invisibly to
// where they are needed, at indeterminate times.
System.out.println(Charset.defaultCharset());