Output, Strings, Input

Simplest Java Program?

```java
public class HelloWorld {
    public static void main(String[] args) {
        System.out.println("Hello World!");
    }
}
```

Simplest Java Program?

```
public class HelloWorld {
  public static void main(String[] args) {
    System                             !");
  }
}
```

public class declares this
code to be a software
component for which
bytecode should be
generated by the compiler;
HelloWorld is the name of
the class;
details later.

Simplest Java Program?

```java
public class HelloWorld {
    public static void main(String[] args) {
        System.out.println("Hello World!");
    }
}
```

public static void is required here when you want a class to include a "main" program that can be executed by the JVM (and it must be called `main`); details later.
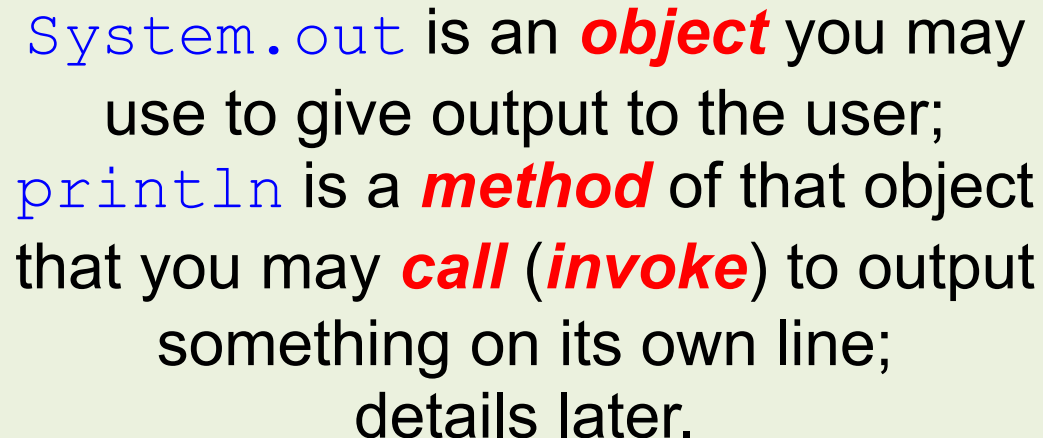
Simplest Java Program?

```java
public class HelloWorld {
    public static void main(String[] args) {
        System.out.println("Hello World!");
    }
}
```

String[] args means that main expects the JVM to hand it an *array* of Strings (called *command-line arguments*) when it is executed; details later.
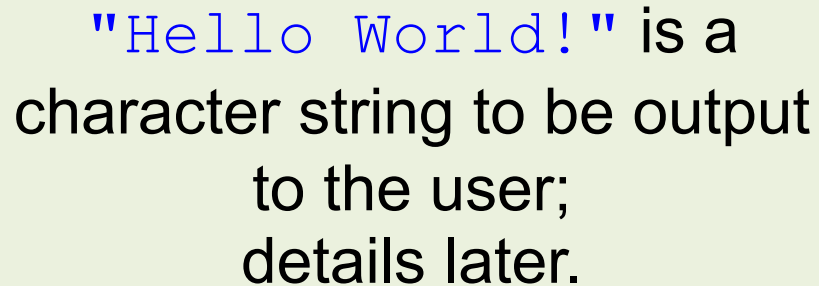
Simplest Java Program?

```java
public class HelloWorld {
    public static void main(String[] args) {
        System.out.println("Hello World!");
    }
}
```

System.out is an ***object*** you may use to give output to the user; println is a ***method*** of that object that you may ***call*** (***invoke***) to output something on its own line; details later.

Simplest Java Program?

```java
public class HelloWorld {
    public static void main(String[] args) {
        System.out.println("Hello World!");
    }
}
```

> "Hello World!" is a character string to be output to the user; details later.

Another Version (sans Comments)

```java
import components.simplewriter.SimpleWriter;
import components.simplewriter.SimpleWriter1L;
public final class HelloWorld {
    private HelloWorld() {
    }
    public static void main(String[] args) {
        SimpleWriter out = new SimpleWriter1L();
        out.println("Hello World!");
        out.close();
    }
}
```

Another Version (sans Comments)

```java
import components.simplewriter.SimpleWriter;
import components.simplewriter.SimpleWriter1L;
public final class HelloWorld {
  private
  }
  public                              s) {
    Simpl                           r1L();
    out.p
    out.c
  }
}
```

> **import** indicates you want to use a component in your code;
> `components` is a ***package*** containing OSU CSE components;
> its `simplewriter` package offers a few advantages over using built-in `System.out`;
> details later.

Another Version (sans Comments)

```java
import components.simplewriter.SimpleWriter;
import components.simplewriter.SimpleWriter1L;
public final class HelloWorld {
  private HelloWorld() {

  }

  public s                        args) {
    SimpleW                       iter1L();
    out.pri
    out.clo
  }
}
```

> **public** means anyone can use this class;
> **final** means no one can incrementally change this class by using ***inheritance***; details later.

Another Version (sans Comments)

```
import components.simplewriter.SimpleWriter;
import components.simplewriter.SimpleWriter1L;
public final class HelloWorld {
    private HelloWorld() {

    }

    public s                              rgs) {
        Simple                    ter1L();;
        out.pr
        out.cl
    }
}
```

> **private** HelloWorld()
> means the HelloWorld class
> does not define a *type*, i.e., no
> one can create an object from
> the class HelloWorld because
> it is a *utility class*;
> details later.

Another Version (sans Comments)

```
import comp            leWriter;
import comp            leWriter1L;
public fina
  private H
  }
  public static oid main(String[] args) {
    SimpleWriter out = new SimpleWriter1L();
    out.println("Hello World!");
    out.close();
  }
}
```

> SimpleWriter is the type of
> a newly declared *variable*;
> out is the name of that
> variable;
> details later.

Another Version (sans Comments)

```
import c...                      ...Writer;
import c...                      ...Writer1L;
public f...
   private
   }

   public static void main(St...[] args) {
      SimpleWriter out = new SimpleWriter1L();
      out.println("Hello World!");
      out.close();
   }
}
```

> **new** creates a new object to which the variable `out` is a *reference*; `SimpleWriter1L` is the class whose code should be used when any method of `out` is called; details later.

Another Version (sans Comments)

```java
import components.simplewriter.SimpleWriter;
import components.simplewriter.SimpleWriter1L;
public final
    private
    }
    public st                            args) {
        SimpleWriter out = new SimpleWriter1L();
        out.println("Hello World!");
        out.close();
    }
}
```

> out has a println method, too, nearly identical to that of System.out; details later.

Another Version (sans Comments)

```
import components.simplewriter.SimpleWriter;
import comp                        pleWriter1L;
public fina
    private
    }
    public s                args) {
        SimpleW              SimpleWriter1L();
        out.println("Hello World!");
        out.close();
    }
}
```

> out has a close method as well, and you need to call it when you are done using out;
> details later.

Output: `SimpleWriter`

- The OSU CSE components provide a simple way to provide output to a user via the *console* or a *file*

```
SimpleWriter consoleOut =
    new SimpleWriter1L();
SimpleWriter fileOut =
    new SimpleWriter1L("foo.txt");
```

Output Examples

```
consoleOut.print("Prompt: ");
consoleOut.println();
fileOut.println("A line.");
```

Closing Output

- When you are done ***writing*** output to a `SimpleWriter` ***stream***, you must ***close*** the stream:

  ```
  consoleOut.close();
  fileOut.close();
  ```

Character Strings

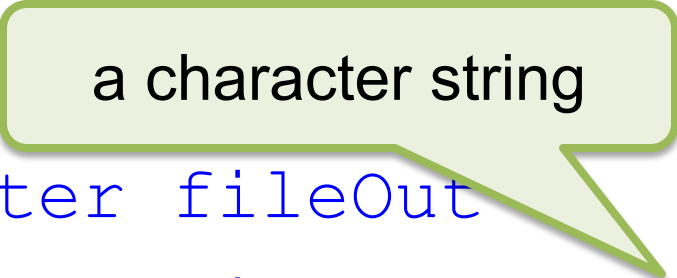- Java has special features to deal with character strings

- Examples

```
SimpleWriter fileOut =
    new SimpleWriter1L("foo.txt");
fileOut.print("Hi, Mr. Foo.");
```

- This intro is just the tip of the iceberg!

Character Strings

- Java has special features to deal with character strings

- Examples

a character string

```
SimpleWriter fileOut
   new SimpleWriter1L("foo.txt");
fileOut.print("Hi, Mr. Foo.");
```

- This intro is just the tip of the iceberg!

Character Strings

- Java has special features to deal with character strings

- Examples

  a character string

```
SimpleWriter fileOut =
    new SimpleWriter1L("foo.txt");
fileOut.print("Hi, Mr. Foo.");
```

- This intro is just the tip of the iceberg!

Character-String Literals

- Character-string constants, also called ***String literals***, are enclosed in double-quotes, e.g.:

  `"Hello World!"`

- Character strings can be ***concatenated*** (joined together to create new character strings) using the **+** operator, e.g.:

  `"Hello " + "World!"`

String Variables

- You may ***declare*** a ***String variable***, and ***assign*** an initial character-string ***value*** to it, as follows:

```
String cheer = "Go";
```



```
cheer
```

String Variables

- You may assign any other character-string value to the same variable later, e.g.:

```
cheer = cheer + " Bucks!";
```

- Before assignment above:

"Go"

cheer

String Variables

- You may assign any other character-string value to the same variable later, e.g.:

```
cheer = cheer + " Bucks!";
```

- After assignment above:

"Go Bucks!"

cheer

Input: SimpleReader

- The OSU CSE components provide a simple way to get input from a user via the *keyboard* or a *file*

```
SimpleReader keyboardIn =
    new SimpleReader1L();
SimpleReader fileIn =
    new SimpleReader1L("foo.txt");
```

Input Examples

```
String line = keyboardIn.nextLine();
line = fileIn.nextLine();
```

Input Examples

```
String line = keyboardIn.nextLine();
line = fileIn.nextLine();
```

This method, which reads up through and including the next **_line separator_**, and returns everything it reads *except* that next line separator, is really the only method you need to read input from the keyboard and text files.

Closing Input

- When you are done ***reading*** input from a `SimpleReader` ***stream***, you must ***close*** the stream:

  ```
  keyboardIn.close();
  fileIn.close();
  ```

Resources

- Java Tutorials ("Hello World" program)

  - http://docs.oracle.com/javase/tutorial/getStarted/application/index.html

- OSU CSE components API
  (`SimpleWriter`, `SimpleReader`)

  - http://cse.osu.edu/software/common/doc/