Object-Oriented Programming

Basic Overview of Java

Outline

- Brief overview of Java
- Structure of a Java program
- Running through basic programming concepts
- Compiling a Java program
- Looking at API

We will be working with Java in the terminal. Please expect to code through the session. ☺️

How many were able to build and run a “Hello World” program in Java?

Please raise your hand if you have ☺️

Image source: https://www.kindpng.com/imgv/oRomoh_raised-hands-png-transparent-hand-emoji-png-download/

How many reviewed or tried programming in Java? What did you code?

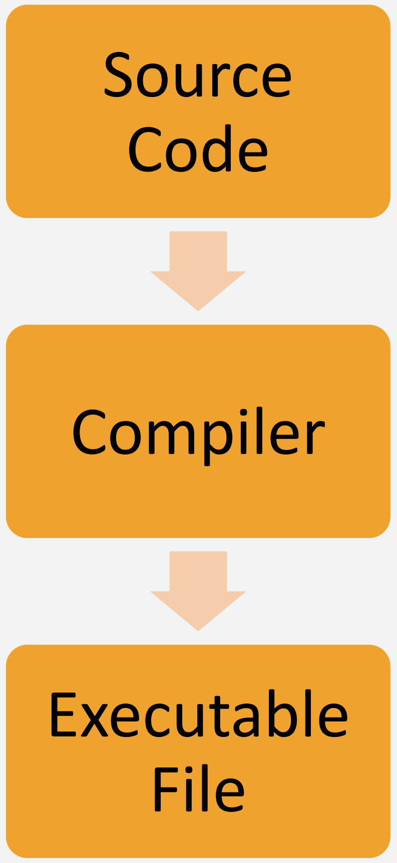
Please raise your hand if you have ☺️

Image source: *https://www.kindpng.com/imgv/oRomoh_raised-hands-png-transparent-hand-emoji-png-download/*

Java

- Is an object-oriented language
    - Object-oriented implies the support of most, if not all, concepts that make up an OO environment
    - Object-based usually refers to the support of the creation of objects but that there's a lack of higher OO concepts, such as inheritance or polymorphism (more on these after the midterm!)
- Is both a compiled and interpreted language
    - What does this mean?

Compiled Language

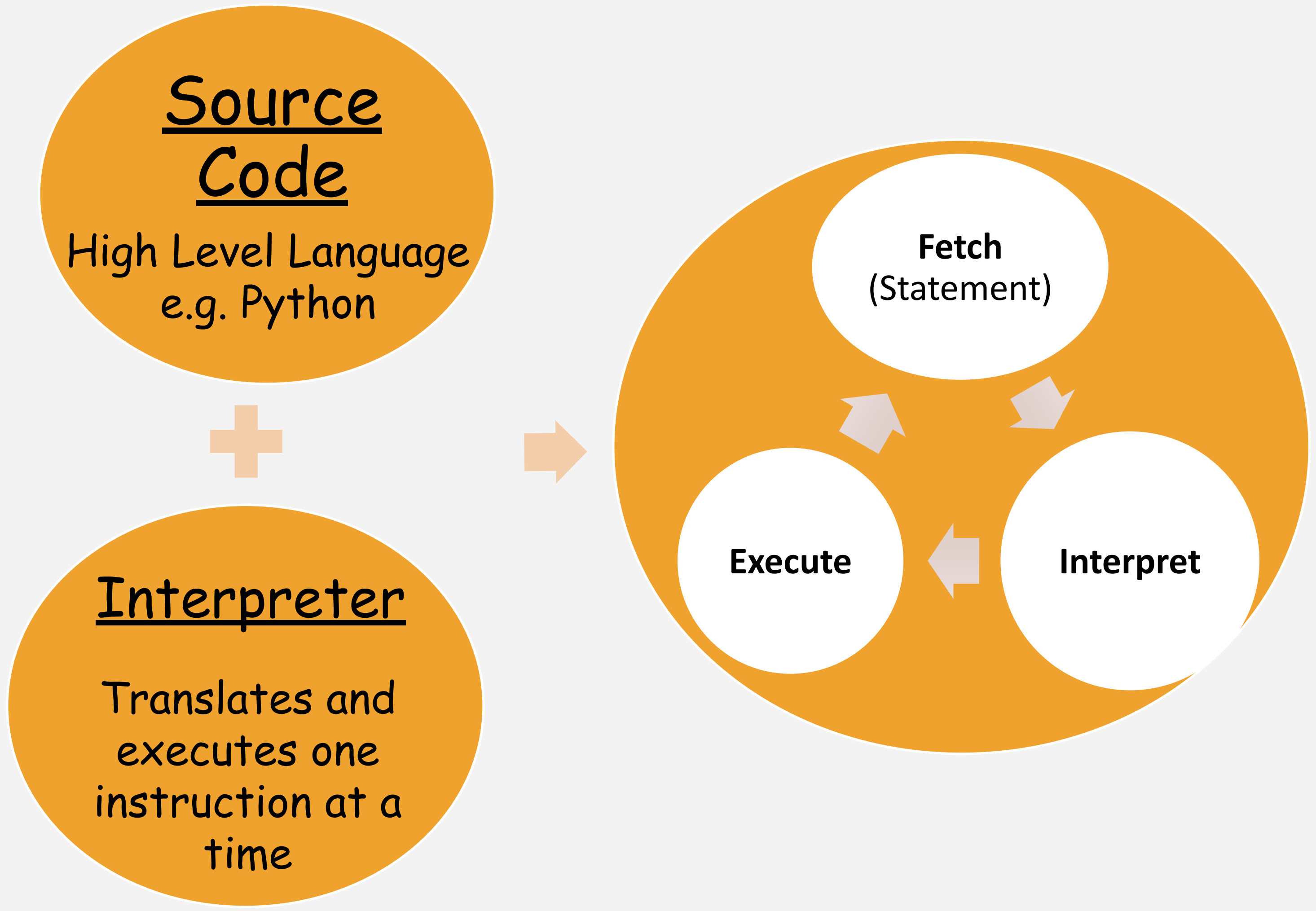


High Level Programming Language
e.g. C, Java

A software program that translates high level language program into an executable machine language program.

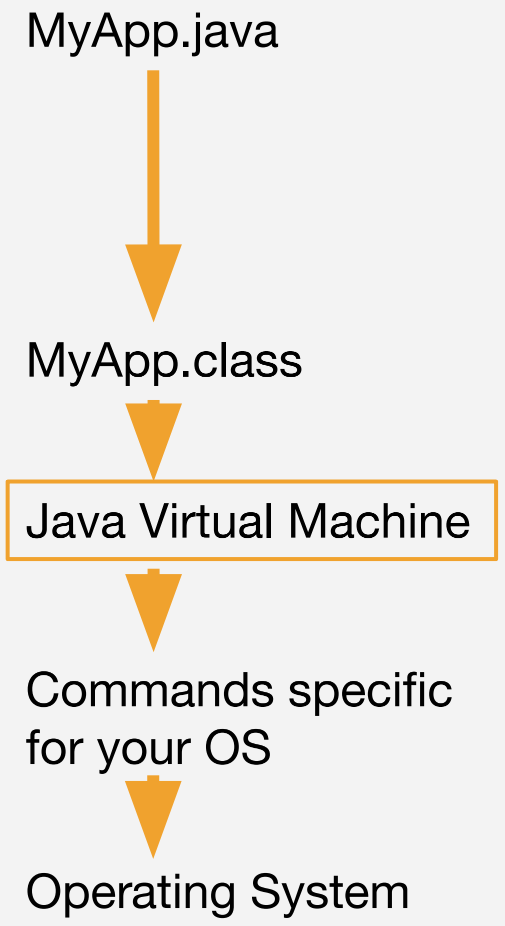
May be executed many times but on one type of computer only

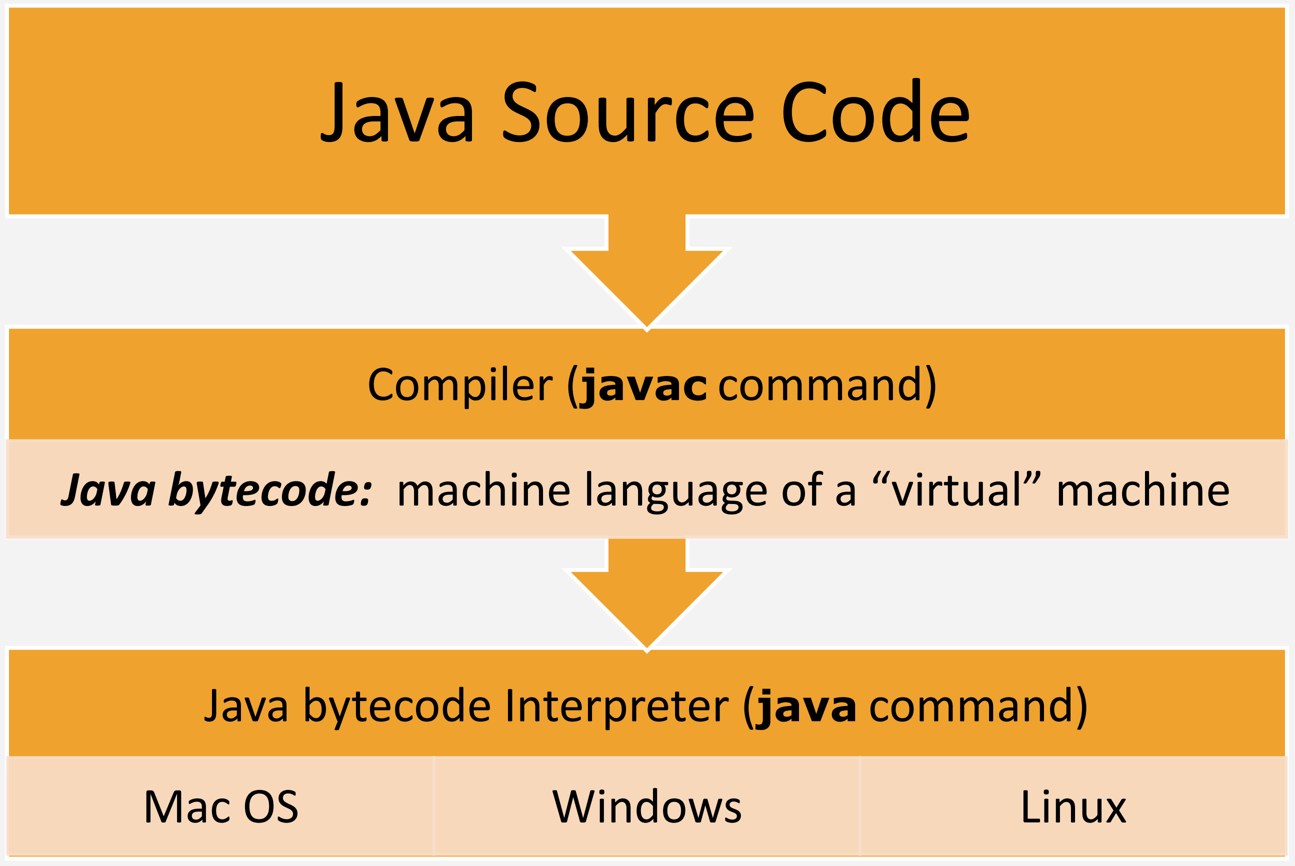
Interpreted Language


Source Code
High Level Language e.g. Python
Interpreter
Translates and executes one instruction at a time
Fetch (Statement)
Interpret
Execute

Java

For example:




Java Source Code
Compiler (**javac** command)
Java bytecode: machine language of a "virtual" machine
Java bytecode Interpreter (**java** command)
Mac OS
Windows
Linux

Writing Java Applications

```
public class MyApp {



}
```

Filename: MyApp.java

- File Extension: .java
- File Name: Class Name
  (exact case & spelling)

Don't worry so much about Access Modifiers for now. We'll discuss more of this in time. ☺️

- A Java application consists of one or more Java classes
- Class declaration for each class:
  - Access modifier (public, private)
  - Class name
    - Should start with an Uppercase Letter
    - CamelCased
  - Braces mark the start and end of the class declaration

Why have Class names in uppercase?

- You don’t have to… but it is a good practice
- Coding standards to follow…
  - Classes start with an uppercase
  - Methods and variables should start with a lowercase
- Apply **snake case** for constants (in all uppercase) *e.g. SNAKE_CASE* and **camel case** *e.g. camelCase* for everything else

Class Declaration

- With at most one **`main()`** method, whose signature is:

  `public static void main (String[] args)`

- The runtime system calls the class's **`main()`** method.

```
public class MyApp {
    public static void main(String[] args) {

    }
}
```

Java language is strongly-typed

- The type of every variable and expression must be known at compile time
  - Primitive Types (boolean, char, int, long, float, double)
  - Reference Types (String, JFrame, user-defined types)
- Indicating the data type of the variable…
  - Limits the type of data and values that the variable can hold
  - Limits the result of an expression

Declaring Variables

- Variables can be declared throughout the code, but they must be declared before first use
    - Name the variable
    - Data type of the variable
- All variable declarations must be within the class or method

Declaring Variables (in methods)

Syntax

We will expand on the syntax when we start looking at classes and objects

```
<dtype> <var1>;
<dtype> <var1>, <var2>;
<dtype> <var1> = <value>;
```

Example

```
int nVal;              // nVal stores an integer value.

double dGrade;         // dGrade is a real number.
char cAnswer, cType;
boolean bStop = false;
```

Output Statement

- To display text on a terminal/console, use
  - `System.out.print()` prints text from wherever the cursor is currently at
  - `System.out.println()` prints text like print() and adds a new line character at the end
  - Take note: Case sensitive!

But wait… what is **<u>System</u>**? Why is there an **<u>out</u>**?

<u>Answer</u>: Consult the <u>API</u>!
https://docs.oracle.com/en/java/javase/12/docs/api/java.base/java/lang/System.html

Output Statement

- To display text on a terminal/console, use
  - `System.out.print()`
  - `System.out.println()`
  - Take note: Case sensitive!
- Parameters placed inside the parenthesis are the values to be displayed on screen
- Parameters may be any expression that would evaluate into a literal

Let’s trace code…

Statements

```
System.out.print("hi");
System.out.print(35);
int nVal = 15;
System.out.print(nVal);
System.out.print(nVal * 3);
```

Screen output

```
hi351545
```

Let’s trace code…

Statements

```
System.out.println("hi");
System.out.print(35);
int nVal = 15;
System.out.println(nVal);
System.out.print(nVal * 3);
```

Screen output

```
hi
3515
45
```

Let’s trace code…

Statements

```
System.out.println("hi\n");
System.out.print("Let’s see\n\nOk?");
```

Screen output

```
hi

Let’s see

Ok?
```

Example

Make sure the filename is the name of the class in exact casing

```
public class MyApp {
   public static void main (String[] args) {
      double dArea = 0.0;
      double dRadius;

      dRadius = 5;
      dArea = 2 * 3.16 * dRadius * dRadius;

      System.out.println("Area is " + dArea);
   }
}
```

Yes… this will work

Let's compile and run this program! ☺️

To compile

```
javac <filename>.java
java <filename>
```

Example

```
javac MyApp.java
java MyApp
```

Looks for a .class file of the specified file name and runs the program

Produces a .class file in your current directory

How was that?
Any issues or problems?

Getting input via the terminal

- It’s not as straight forward as in C or Python…
- We must use a Scanner object to help us get input
- Things we need to do:
  1. Import the Scanner class so our file has access to it
  2. Declare a Scanner variable
  3. Instantiate a Scanner object passing the InputStream
  4. Use the Scanner object to get input
  5. Close the Scanner’s input stream

Importing…

- The Java API is an extensive library… so much that it isn’t good to have everything accessible right away
  - Importing everything from multiple libraries might lead to conflicting imports
- Regardless, we’ll need to import classes that are useful in helping us solve problems

Importing… Scanner

OR java.util.*; which imports all classes under java.util

```
import java.util.Scanner;

public class MyApp {
   public static void main (String[] args) {
      double dArea = 0.0;
      double dRadius;

      dRadius = 5;
      dArea = 2 * 3.16 * dRadius * dRadius;

      System.out.println("Area is " + dArea);
   }
}
```

API of Scanner:
https://docs.oracle.com/en/java/javase/12/docs/api/java.base/java/util/class-use/Scanner.html

Declaring a Scanner Object…

```
import java.util.Scanner;

public class MyApp {
   public static void main (String[] args) {
      Scanner sc;

      double dArea = 0.0;
      double dRadius;

      dRadius = 5;
      dArea = 2 * 3.16 * dRadius * dRadius;

      System.out.println("Area is " + dArea);
   }
}
```

Scanner

- Is a reference data type
- Reference data types need to be instantiated before use
- I.e. we need to create an instance of a class
- Needs some input stream which can be…
  - From the terminal
  - From a file

Instantiating a Scanner object...

```
import java.util.Scanner;

public class MyApp {
   public static void main (String[] args) {
      Scanner sc = new Scanner(System.in);

      double dArea = 0.0;
      double dRadius;

      dRadius = 5;
      dArea = 2 * 3.16 * dRadius * dRadius;

      System.out.println("Area is " + dArea);
   }
}
```

Scanner

- As we just want to get input from the terminal, we can use the InputStream of System and pass that in as a parameter

This is a form of abstract in code as we can somewhat understand that Scanner does something with the input stream, but we don't see the code

Getting input...

```
import java.util.Scanner;

public class MyApp {
   public static void main (String[] args) {
      Scanner sc = new Scanner(System.in);

      double dArea = 0.0;
      double dRadius;

      dRadius = sc.nextDouble();
      dArea = 2 * 3.16 * dRadius * dRadius;

      System.out.println("Area is " + dArea);
   }
}
```

Scanner

- As we just want to get input from the terminal, we can use the InputStream of System and pass that in as a parameter
- nextDouble() returns a double value that was input by the user
- Check the API for other Scanner methods

In our modification here, we assume the radius is unknown and that the user should give it as an input.

Closing the Scanner’s input stream…

```
import java.util.Scanner;

public class MyApp {
   public static void main (String[] args) {
      Scanner sc = new Scanner(System.in);

      double dArea = 0.0;
      double dRadius;

      dRadius = sc.nextDouble();
      dArea = 2 * 3.16 * dRadius * dRadius;

      System.out.println("Area is " + dArea);

      sc.close();
   }
}
```

Good Coding Practice

- When done with an InputStream, close it to avoid a resource leak
- If you still plan to use it, there’s no need to close it
- Just don’t forget to close it once you’re done

- Although… if you do forget to close it, the JVM will eventually collect it. It is just good to practice closing your streams because InputStream can come from different places.

Comments in Java

```
// line comment

/* multi-line
   comment */


/** javadoc
    comment
 */
```

- Internal documentation
  - Notes within your Java code
- Types of Comments
  - Line Comment
    - Starts with //
    - Texts in the same line after // will be disregarded by the compiler
  - Multi-line Comment
    - Enclosed within /* and */
    - Texts within the markers are disregarded
  - Javadoc Comment
    - Enclosed within /** and */
    - How to write javadoc comments: https://www.oracle.com/ph/technical-resources/articles/java/javadoc-tool.html

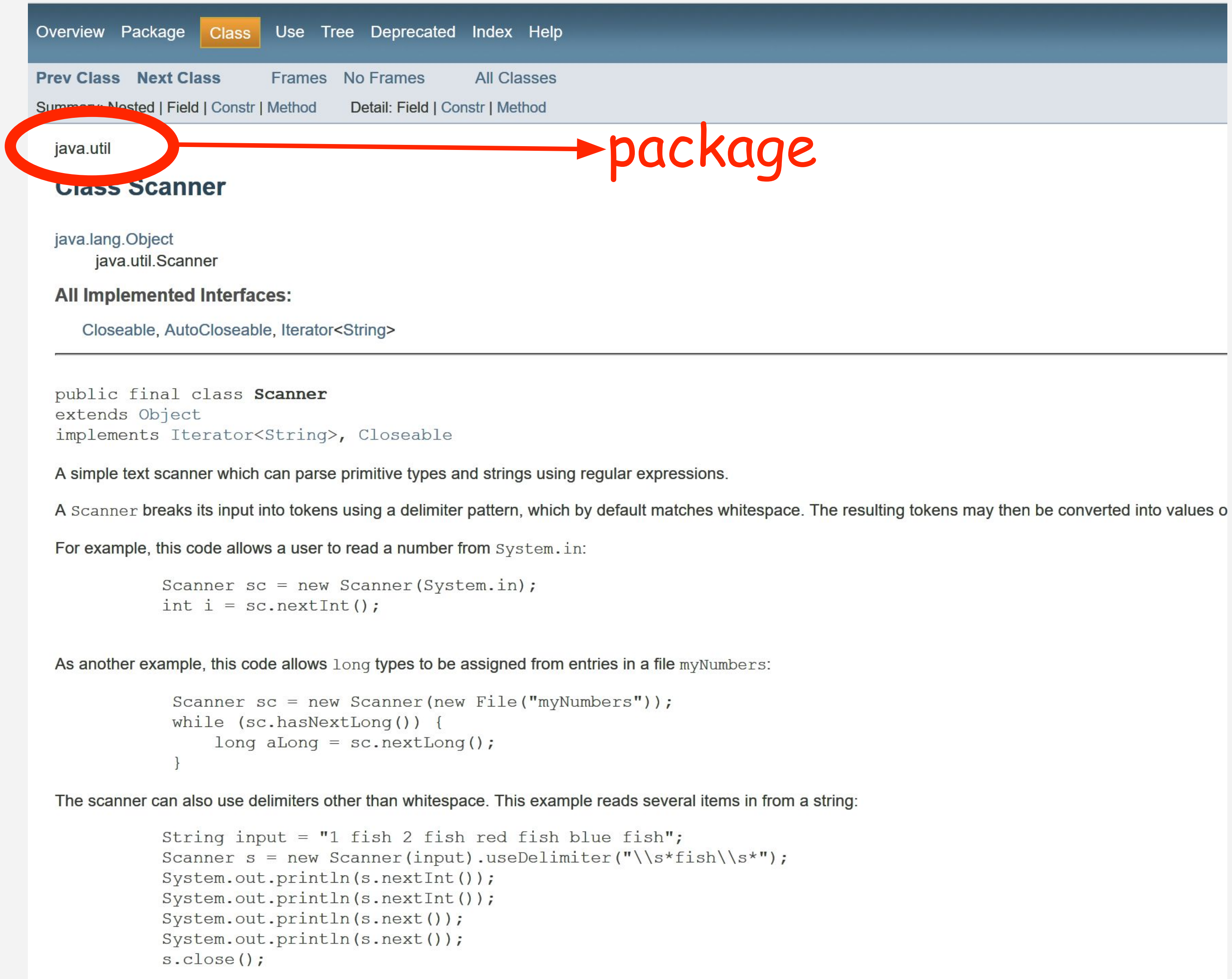
Javadoc is going to be important to understand so you can generate your own system's documentation

Example

```
/**   This is my first Java application!
  *   @author  Ed
  */
public class MyApp {
   public static void main (String[] args) {
       /* Multi-line
        * Comment
        */
       // Single-line comment
   }
}
```

You'll need to get use to reading Java's API Documentation

API Documentation


Overview Package Class Use Tree Deprecated Index Help
Prev Class Next Class Frames No Frames All Classes
Summary: Nested | Field | Constr | Method Detail: Field | Constr | Method
package

java.util

Class Scanner

java.lang.Object
java.util.Scanner

All Implemented Interfaces:

Closeable, AutoCloseable, Iterator<String>

```
public final class Scanner
extends Object
implements Iterator<String>, Closeable
```

A simple text scanner which can parse primitive types and strings using regular expressions.

A `Scanner` breaks its input into tokens using a delimiter pattern, which by default matches whitespace. The resulting tokens may then be converted into values o

For example, this code allows a user to read a number from `System.in`:

```
Scanner sc = new Scanner(System.in);
int i = sc.nextInt();
```

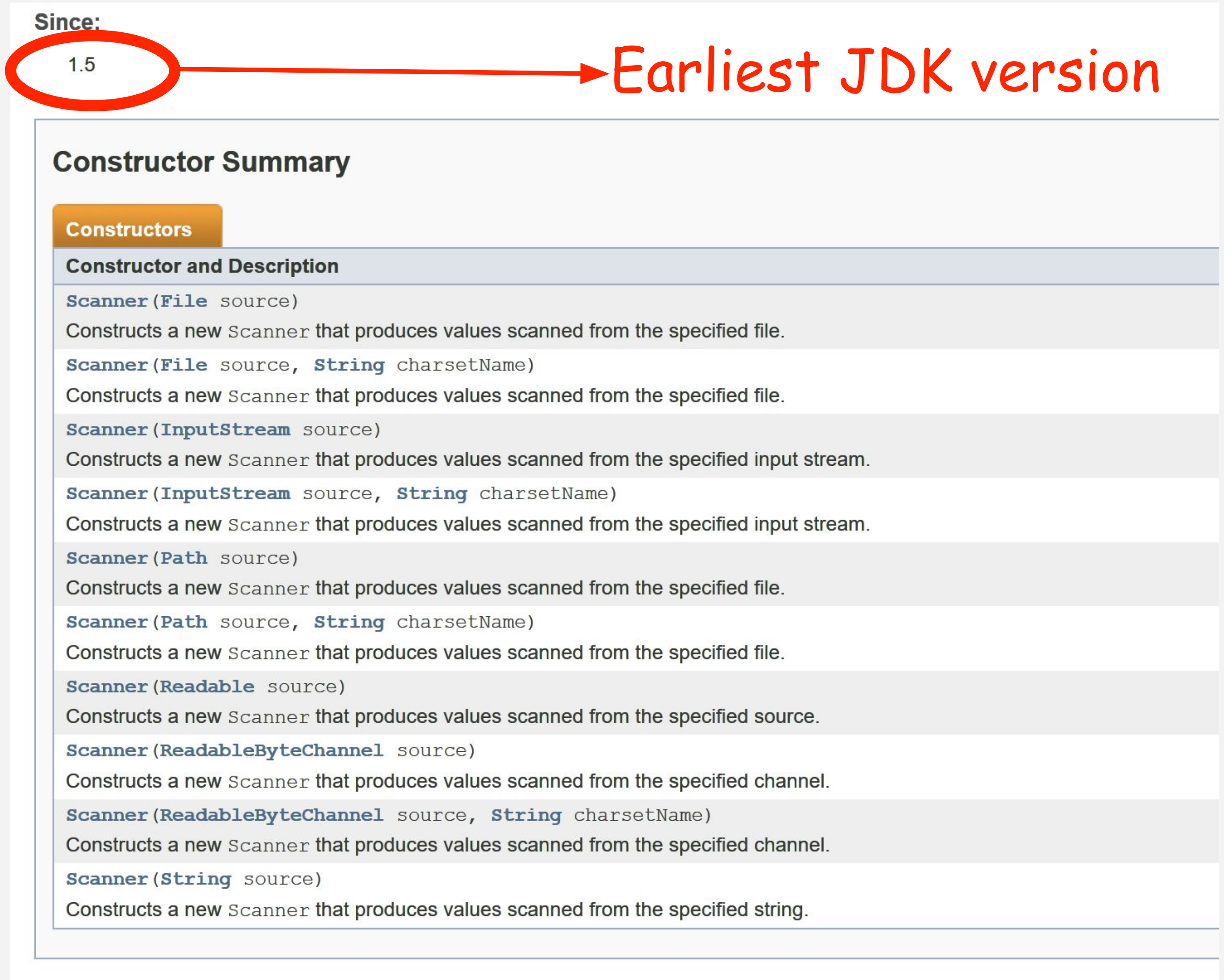
As another example, this code allows `long` types to be assigned from entries in a file `myNumbers`:

```
 Scanner sc = new Scanner(new File("myNumbers"));
 while (sc.hasNextLong()) {
     long aLong = sc.nextLong();
 }
```

The scanner can also use delimiters other than whitespace. This example reads several items in from a string:

```
String input = "1 fish 2 fish red fish blue fish";
Scanner s = new Scanner(input).useDelimiter("\\s*fish\\s*");
System.out.println(s.nextInt());
System.out.println(s.nextInt());
System.out.println(s.next());
System.out.println(s.next());
s.close();
```

API Documentation



Since:

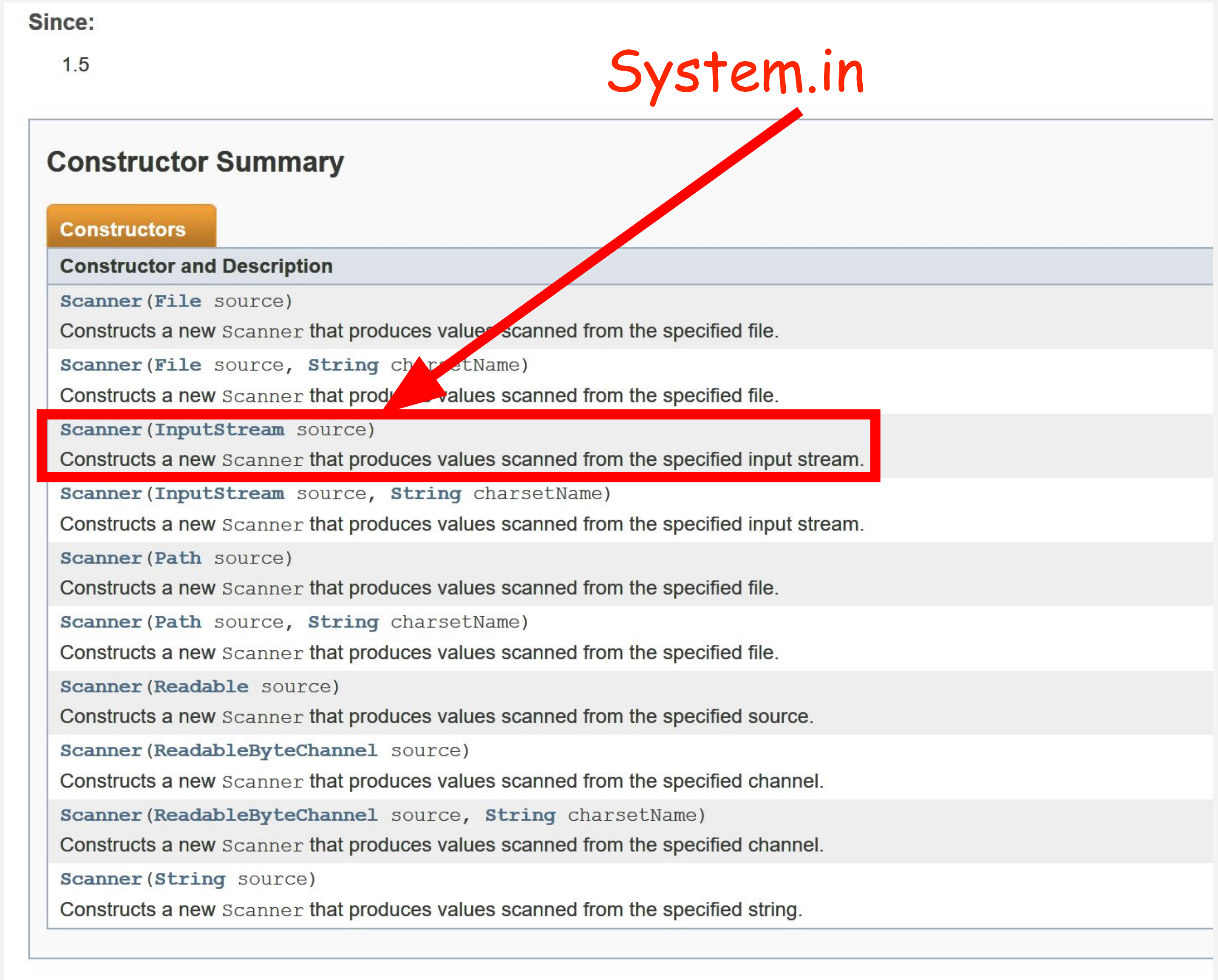
1.5

Constructor Summary

Constructors

Constructor and Description
`Scanner(File source)` Constructs a new `Scanner` that produces values scanned from the specified file.
`Scanner(File source, String charsetName)` Constructs a new `Scanner` that produces values scanned from the specified file.
`Scanner(InputStream source)` Constructs a new `Scanner` that produces values scanned from the specified input stream.
`Scanner(InputStream source, String charsetName)` Constructs a new `Scanner` that produces values scanned from the specified input stream.
`Scanner(Path source)` Constructs a new `Scanner` that produces values scanned from the specified file.
`Scanner(Path source, String charsetName)` Constructs a new `Scanner` that produces values scanned from the specified file.
`Scanner(Readable source)` Constructs a new `Scanner` that produces values scanned from the specified source.
`Scanner(ReadableByteChannel source)` Constructs a new `Scanner` that produces values scanned from the specified channel.
`Scanner(ReadableByteChannel source, String charsetName)` Constructs a new `Scanner` that produces values scanned from the specified channel.
`Scanner(String source)` Constructs a new `Scanner` that produces values scanned from the specified string.

API Documentation



Since:

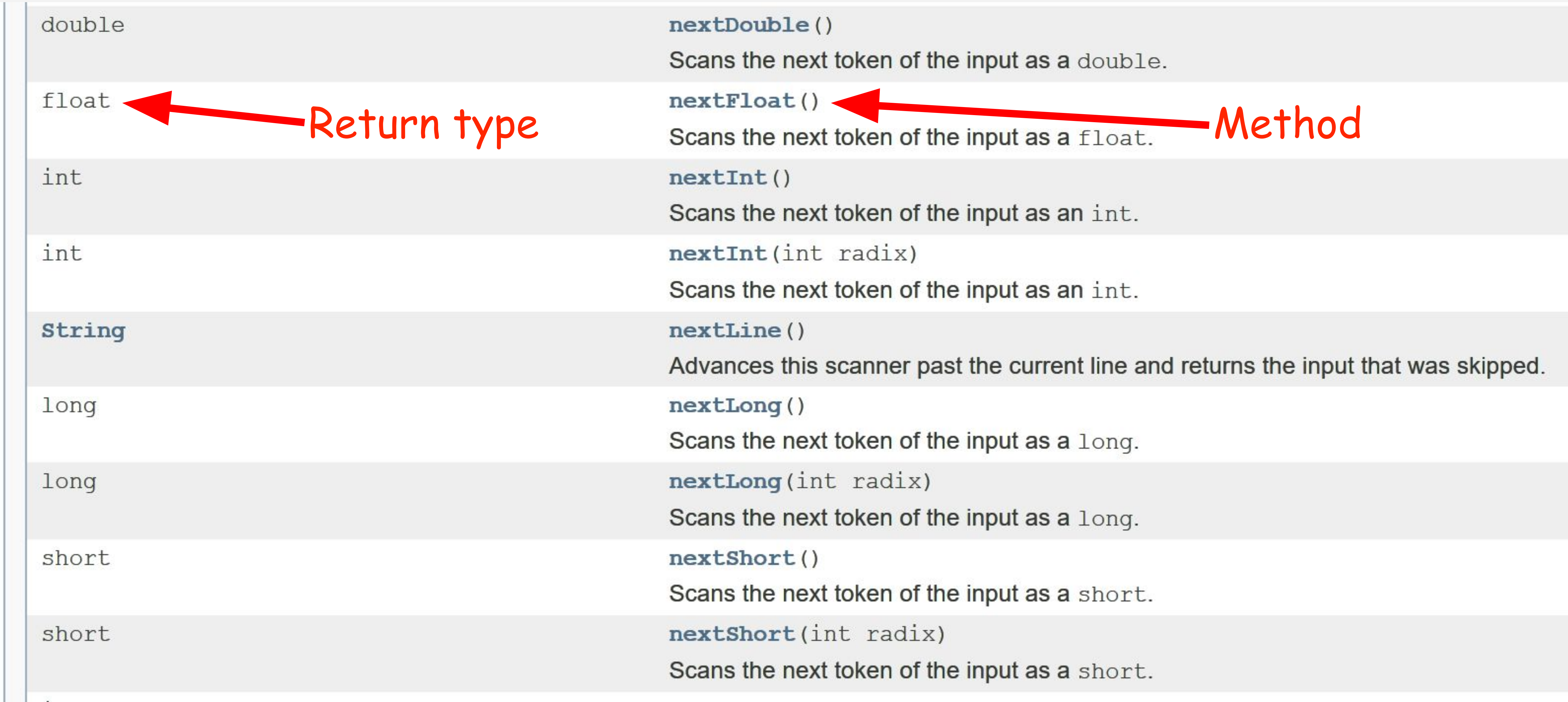
1.5

Constructor Summary

Constructors

Constructor and Description
`Scanner(File source)` Constructs a new `Scanner` that produces values scanned from the specified file.
`Scanner(File source, String charsetName)` Constructs a new `Scanner` that produces values scanned from the specified file.
`Scanner(InputStream source)` Constructs a new `Scanner` that produces values scanned from the specified input stream.
`Scanner(InputStream source, String charsetName)` Constructs a new `Scanner` that produces values scanned from the specified input stream.
`Scanner(Path source)` Constructs a new `Scanner` that produces values scanned from the specified file.
`Scanner(Path source, String charsetName)` Constructs a new `Scanner` that produces values scanned from the specified file.
`Scanner(Readable source)` Constructs a new `Scanner` that produces values scanned from the specified source.
`Scanner(ReadableByteChannel source)` Constructs a new `Scanner` that produces values scanned from the specified channel.
`Scanner(ReadableByteChannel source, String charsetName)` Constructs a new `Scanner` that produces values scanned from the specified channel.
`Scanner(String source)` Constructs a new `Scanner` that produces values scanned from the specified string.

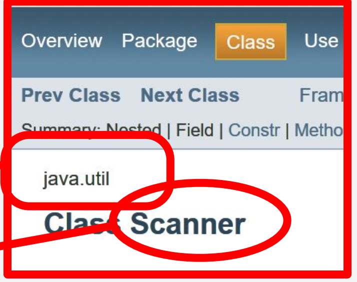
API Documentation



double	**nextDouble**() Scans the next token of the input as a `double`.
float	**nextFloat**() Scans the next token of the input as a `float`.
int	**nextInt**() Scans the next token of the input as an `int`.
int	**nextInt**(int radix) Scans the next token of the input as an `int`.
String	**nextLine**() Advances this scanner past the current line and returns the input that was skipped.
long	**nextLong**() Scans the next token of the input as a `long`.
long	**nextLong**(int radix) Scans the next token of the input as a `long`.
short	**nextShort**() Scans the next token of the input as a `short`.
short	**nextShort**(int radix) Scans the next token of the input as a `short`.

```
import java.util.*;

public class ScannerTest {
   public static void main (String[] args) {
      Scanner sc = new Scanner (System.in);
```



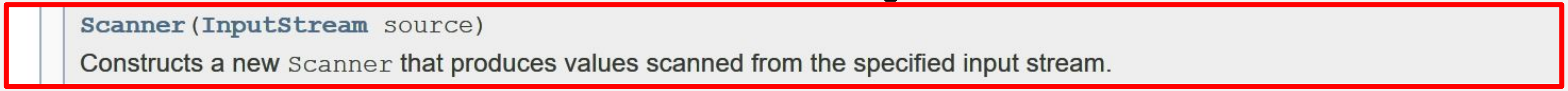
Overview Package Class Use

Prev Class Next Class Fram

Summary: Nested | Field | Constr | Metho

java.util

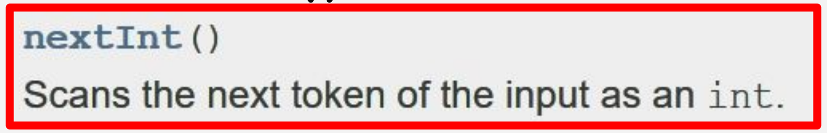
Class Scanner



`Scanner(InputStream source)`

Constructs a new `Scanner` that produces values scanned from the specified input stream.

```
      System.out.print (“How old are you? ”);
      int nAge = sc.nextInt ();
```

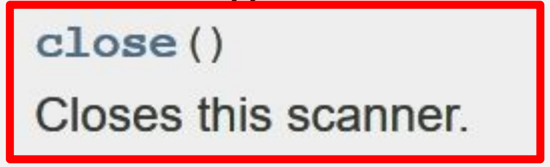


`nextInt()`

Scans the next token of the input as an `int`.

```
      int nYrOfBirth = 2019 - nAge;

      System.out.println (“Approx. year of birth: ” +
                           nYrOfBirth);
      sc.close ();
   }
}
```



`close()`

Closes this scanner.

Questions so far?

Question

How does Java provide an environment for object-oriented programming?

Summary

- Java provides developers with an object-oriented environment
    - Forces you to think of a solution that makes use of objects
- A Java application can have one or more classes
    - Running a Java app implies running the main() method
- Many programming constructs are the same with C, but there are some difference

Summary

- Documentation is an important skill to develop
  - Forces you to explain your code
  - Helps others understand your code
- There are a lot of classes that are part of the Java API
  - Some are immediately accessible, while others are not
  - Get use to consulting API/documentation

Next meeting…

- Have a graded exercise

Practice Exercises:

- A couple of problems while working with Java
    - Working with Arrays
    - String Equivalences

Keep learning…