Inf1-OP

Classes and Objects - Part III

Volker Seeker, adapting earlier version by Perdita Stevens and Ewan Klein

School of Informatics

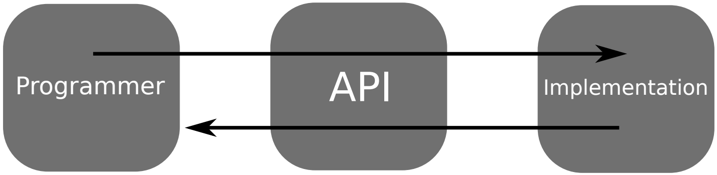
February 5, 2019

Built-in Classes

The Java API / Class Library

Application Programming Interface

The interface between the user of the code and the implementation itself is called an Application Programming Interface (API).



Major Benefit: Underlying implementation can be changed (improved) without affecting the user of the API.

Java API

Some functionality is used often by most programs, e.g.

- Printing to the console: `System.out.println("Hi")`
- Handling sequences of multiple characters:
  `String msg = "Error: invalid value!"`
- Generating a random number:
  `Integer num = Integer.parseInt(args[0])`
- etc.

To avoid the reinvention of the wheel over and over, a library with standard functionality and classes is provided for every programming language

In Java this is called the **Java API** or **Java Documentation**
http://docs.oracle.com/javase/8/docs/api/

Packages

Organising Classes

Organising code

Things that need to be changed together should live together.

But **Classes** are not enough.

Organising code

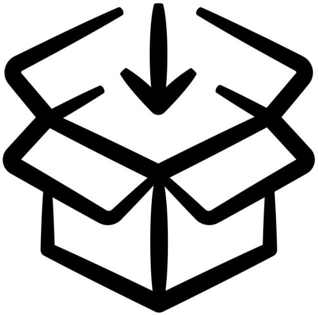
Java Version	**Number of Classes in Library**
11	4410
10	6002
9	6005
8	4240
7	4024
6	3793
5.0	3279
1.4.2	2723
1.3.1	1840

Organising code

Java Version	Number of Classes in Library
11	4410
10	6002
9	6005
8	4240
7	4024
6	3793
5.0	3279
1.4.2	2723
1.3.1	1840

A way of organising code on a higher level is needed, i.e. of organising classes.

Organising classes in packages



Created by Gregor Cresnar
from Noun Project

In Java, **packages** are used to organise classes.

Think of them as subfolders (which they usually are anyway).

Organising classes in packages

Consider for example `java.lang` which contains fundamental classes for using the language, e.g. `Integer`, `Maths`, `String`

or

`java.util` which contains various utility classes, e.g. `Arrays`, `Date`, `Scanner`

Naming Convention package names start with a lower case symbol and subpackages separated by '.'

Using Packages

Using a classes from a package in your code, requires you to specify the entire name including the package prefix:

```
public class DatePrinter {
   public static void main(String[] args) {
        java.util.Date today = new java.util.Date();
        System.out.println("Today's date is: "
                + today.toString());
   }
}
```

Output

```
Today's date is: Thu Jan 31 09:34:09 GMT 2019
```

Using Packages

To save you some writing work, you can import necessary classes.
This allows you to skip the package prefix.

```
import java.util.Date;
public class DatePrinter {
    public static void main(String[] args) {
         Date today = new Date();
         System.out.println("Today's date is: "
                 + today.toString());
    }
 }
```

Import statements need to be outside of the class definition.

You can also import all classes from a package:

`import java.util.*`

Using Packages

I am using `Integer`, `String` and `Maths` all the time but never need to import anything!

Using Packages

I am using `Integer`, `String` and `Maths` all the time but never need to import anything!

All classes from the **java.lang** package are included automatically into every Java program.

Creating your own packages

You can create your own packages by using the package keyword.

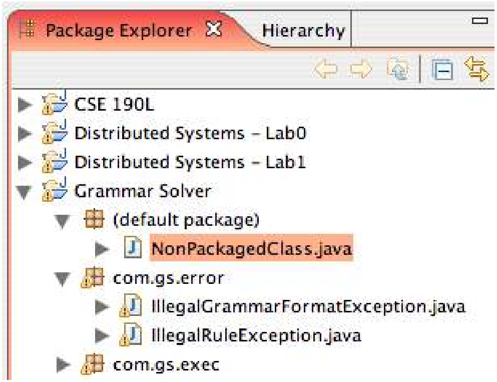
```
package com.dateapp.output;

import java.util.Date;
public class DatePrinter {
    public static void main(String[] args) {
         Date today = new Date();
         System.out.println("Today's date is: "
                 + today.toString());
    }
 }
```

The package definition needs to go into the first line of your class document.

Also, make sure you put the underlying file in the correct subfolder.

Default package

The **default package** indicates that your source files are in no particular package.


Package Explorer
Hierarchy
CSE 190L
Distributed Systems - Lab0
Distributed Systems - Lab1
Grammar Solver
(default package)
NonPackagedClass.java
com.gs.error
IllegalGrammarFormatException.java
IllegalRuleException.java
com.gs.exec

source: https://courses.cs.washington.edu/courses/cse143/19wi/eclipse_tutorial/packages.shtml

Namespace management

Packages maintain their own isolated namespaces

`com.myapp.graphics.Utils`

`com.myapp.io.Utils`

Classes with the same name can co-exist in the same program if they are in different packages.

Java API

With this knowledge, lets take another quick look at the API.

`http://docs.oracle.com/javase/8/docs/api/`

Strings

An example from the class library

String: basis for text processing

Underlying set of values: sequences of Unicode characters.

`public class String`

	`String(String s)`	*create a string with same value as* s
`char`	`charAt(int i)`	*character at index* i
`String`	`concat(String t)`	*this string with* t *appended*
`int`	`compareTo(String t)`	*compare lexicographically with* t
`boolean`	`endsWith(String post)`	*does string end with* `post`*?*
`boolean`	`equals(Object t)`	*is* t *a* `String` *equal to* `this` *one?*
`int`	`indexOf(String p)`	*index of first occurrence of* p
`int`	`indexOf(String p, int i)`	*as* `indexOf`*, starting search at index* i
`int`	`length()`	*return length of string*
`String`	`replaceAll(String a, String b)`	*result of changing all as to bs*
`String[]`	`split(String delim)`	*result of splitting string at* `delim`
`boolean`	`startsWith(String pre)`	*does string start with* `pre`*?*
`String`	`substring(int i, int j)`	*from index* i *to index* $j-1$ *inclusive*

http://docs.oracle.com/javase/8/docs/api/java/lang/String.html

Typical String Processing Code

is the string a palindrome?

```
public static boolean isPalindrome(String s) {
    int N = s.length();
    for (int i = 0; i < N / 2; i++) {
        if (s.charAt(i) != s.charAt(N - 1 - i))
            return false;
    }
    return true;
}
```

---

extract filenames and extensions from a command-line argument

```
String s = args[0];
int dot = s.indexOf(".");
String base = s.substring(0, dot);
String extension = s.substring(dot + 1, s.length());
```

---

print all lines from standard input containing the string "info"

```
while (!StdIn.isEmpty()) {
    String s = StdIn.readLine();
    if (s.contains("info"))
        System.out.println(s);
}
```

---

print all $ac.uk$ *URLs in text file on standard input*

```
while (!StdIn.isEmpty()) {
    String s = StdIn.readString();
    if (s.startsWith("http://") && s.endsWith("ac.uk
        System.out.println(s);
}
```

Format Strings

How to gain more fine-grained control over print strings.

println can be Clunky

The student named 'Lee' is aged 18.

Using string concatenation

```
System.out.println("The student named '"
                   + name
                   + "' is aged "
                   + age
                   + ".");
```

String with Format Specifiers, 1

Target String

```
"The student named 'Lee' is aged 18."
```

String with Format Specifiers, 1

Target String

```
"The student named 'Lee' is aged 18."
```

String with Gaps

```
"The student named '_' is aged _."
```

String with Format Specifiers, 1

Target String

```
"The student named 'Lee' is aged 18."
```

String with Gaps

```
"The student named '_' is aged _."
```

String with Format Specifiers

```
"The student named '%s' is aged %s."
```

String with Format Specifiers, 1

Target String

```
"The student named 'Lee' is aged 18."
```

String with Gaps

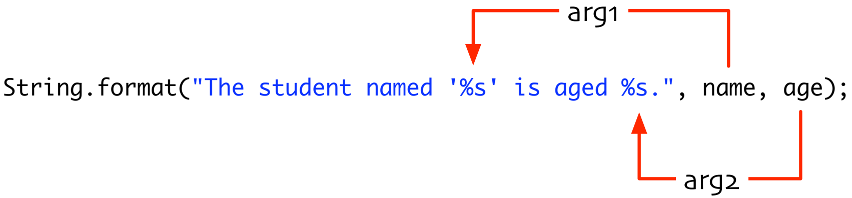
```
"The student named '_' is aged _."
```

String with Format Specifiers

```
"The student named '%s' is aged %s."
```

- %s is a placeholder for a string.
- Called a format specifier.
- Each format specifier in a string gets replaced by an actual value.

String with Format Specifiers, 2



```
String.format("The student named '%s' is aged %s.", name, age);
```

String with Format Specifiers, 3

Define a Format String

```
String str =
    String.format("The student named '%s' is aged %s.",
                  name, age);
System.out.println(str);
```

Output

```
The student named 'Lee' is aged 18.
```

printf, 1

Shorter version

```
System.out.printf("The student named '%s' is aged %s.",
                  name, age);
```

Output

```
The student named 'Lee' is aged 18.
```

printf, 2

Convert char to String

```
System.out.printf("'%s' is for Apple.", 'A');
```

Output

```
'A' is for Apple.
```

printf, 2

Round to 2 decimal places

```
System.out.printf("The value of pi is %f", Math.PI);
System.out.printf("The value of pi is %.2f", Math.PI);
```

Output

```
The value of pi is 3.141593
The value of pi is 3.14
```

printf, 2

Round to 2 decimal places

```
System.out.printf("The value of pi is %f", Math.PI);
System.out.printf("The value of pi is %.2f", Math.PI);
```

Output

```
The value of pi is 3.141593
The value of pi is 3.14
```

Include a newline

```
System.out.printf("The value of pi is %f\n", Math.PI);
```

Summary

- The Java language comes with a set of predefined classes wrapping up most often used functionality.
- Packages are used to organise classes by topic.
- Strings and String formatting are useful

Reading

Java Tutorial

Chapter 8 *Packages*

Chapter 9 *Numbers and Strings*