Class Libraries and Packages

Wolfgang Schreiner
Research Institute for Symbolic Computation (RISC)
Johannes Kepler University, Linz, Austria

Wolfgang.Schreiner@risc.jku.at
http://www.risc.jku.at

Class Libraries

- Class library: a set of classes supporting program development.
  - Java API: class library shipped with every Java system.
  - Class libraries are grouped into several packages.
- Package: a set of classes with related functionality.
  - Some 50 packages in the Java API.

`java.applet`	Create applets
`java.awt`	Abstract Windowing Toolkit
`java.io`	Input and output functions
`java.lang`	General language support
`java.net`	Computing across the network
`java.util`	General utilities

https://docs.oracle.com/en/java/javase/19/docs/api/

Class Names

A class name is only unique within a package.

- Classes in `java.lang` are automatically available.
  - Extension of the Java language.
  - Example: `String`, `Integer`, ...
- Classes in other packages can be referenced by qualified names:
  - *`package.Class`*
  - Example: `java.util.Vector`

Typical: import of classes from other packages.

Class Import

- Classes can be imported into a class declaration.
  - Class name can be used without qualification by package name.

```
import package.Class;
import package.*;
class MyClass
{
  ...
}
```

- `import package.Class;`
  - *Class* can be used without qualification.
  - `import java.util.Vector;`
- `import package.*;`
  - All classes of *package* can be used without qualification.
  - `import java.util.*;`

Writing Packages

Place all classes of a package into a subdirectory of the same name.

```
/home/ws/
  Main.java
  cs1/
    Input.java
    A.java
```

- Package `cs1`.
  - Classes `Input` and `A`.
  - Compilation: `javac cs1/Input.java`
- Usage:

```
import cs1.*;
class Main
{ ... }
```

Classes in Packages

Classes in packages must be especially declared.

```
package cs1;
import ...;
public class Input
{ ... }
```

- Prefix `package` *`package`*`;`
  - Class belongs to *package*.
- Keyword `public` exports the class.
  - Class can be used from classes in other packages.
  - Without `public`, class can be only used within package.

Place your program classes into a separate package.

Nested Packages

Packages may be arbitarily nested.

```
/home/ws/
  Main.java
  cs1/
    assert/
      A.java
```

- Package `cs1.assert`
  - Declaration:

    ```
    package cs1.assert;
    public class A { ... }
    ```

  - Compilation: `javac cs1/assert/A.java`
  - Usage: `import cs1.assert.*;`

The Package `java.lang`

Extension of the Java language.

`Boolean`	A wrapper for type `boolean`
`Byte`	A wrapper for type `byte`
`Character`	A wrapper for type `char` and utility methods
`Double`	A wrapper for type `double` and utility methods
`Float`	A wrapper for type `float` and utility methods
`Integer`	A wrapper for type `int` and utility methods
`Long`	A wrapper for type `long` and utility methods
`Math`	Various numerical (floating point) operations
`Short`	A wrapper for type `short`
`String`	Sequences of characters
`System`	Several useful class fields and class methods

Study the classes in this package in detail.

Example

`System.out.println(`*`string`*`)`

- Class `System` in package `java.lang`
  - Class name is automatically imported.
- Class variable `out` in class `System`

  ```
  static PrintStream out
  ```

  - Class `PrintStream` is in package `java.io`.
- Object method `println` in class `PrintStream`.

  ```
  void println(String)
  ```

Use the hypertext manual to look up the definitions.

Wrapper Classes

For every primitive type, there is a "wrapper" class.

```
int val = 7;
Integer i = new Integer(val);
int value = i.intValue();
```

- Wraps an `int` value into an `Integer` object.
- Extracts the `int` value from the `Integer` object.
- Autoboxing: automatic wrapping/unwrapping when needed.
  - When an object pointer rather than an atomic value is needed.
  - Thus the use of wrapper classes is often hidden from the user.

Other: `Byte`, `Short`, `Long`, `Character`, `Float`, `Double`, `Boolean`.

Example: Separating Words

We write a program that reads a string which contains multiple words (sequences of letters) separated by other characters. The program then prints all words of the text in separate lines in the order of their occurrence in the text. For instance, the input

```
One, two, and three!
```

shall result in output

```
One
two
and
three
```

For this purpose, we may use the `Character` method

```
public static boolean isLetter(char ch)
```

which returns true if and only if character ch denotes a letter.

Main Method

```
public static void main(String[] args)
{
  String text = Input.readString();
  // ... error check omitted
  printWords(text);
}

// printWords(text)
// prints each word of text on a separate line
// Input: text is not null
// Effect: let a word be a sequence of letters.
//   This method prints all word of text in the
//   order of their occurrences on separate lines.
public static void printWords(String text)
```

Core Method

```
public static void printWords(String text)
{
  int i = 0;
  final int end = text.length();
  while (i < end)
  {
    int a = wordBegin(text, i);
    assert(a == end || Character.isLetter(text.charAt(a)));
    if (a == end) break;
    int b = wordEnd(text, a);
    assert(b == end || !Character.isLetter(text.charAt(b)));
    System.out.println(text.substring(a, b));
    i = b+1;
  }
}
```

Specification of Auxiliary Methods

```
// p = wordBegin(text, i)
// p is the position of the first letter at or after i
// Input: text is not null, i < length(text)
// Output: i <= p <= length(text)
//   text at j is not a letter for every i <= j < p.
//   if p < length(text), then text at p is a letter.
static int wordBegin(String text, int i)

// p = wordEnd(text, i)
// p is the position of the first non-letter after i
// Input: text is not null, i < length(text)
// Output: i < p <= length(text)
//   text at j is a letter for every i < j < p.
//   if p < length(text), then text at p is not a letter.
static int wordEnd(String text, int i)
```

Implementation of Auxiliary Methods

```
static int wordBegin(String text, int i)
{ int a = i;
  final int end = text.length();
  while (a < end && !Character.isLetter(text.charAt(a)))
    a = a+1;
  return a;
}

static int wordEnd(String text, int a)
{ int b = a+1;
  final int end = text.length();
  while (b < end && Character.isLetter(text.charAt(b)))
    b = b+1;
  return b;
}
```