Voortgezet Programmeren

Lecture 2: Programming with Java

Tommi Tervonen

Econometric Institute, Erasmus University Rotterdam

Packages in Java

- Multiple classes can have same name, as long as they are in different packages
- Same package classes are automatically in the same namespace
- Others you need to import or refer to them explicitly (java.util.ArrayList)
- Some standard library packages:
  - `java.lang` (core classes, always in the namespace)
  - `java.util`
  - `java.io`
  - `java.math`
- Convention: name package according to domain in inverse order (`fi.smaa.jsmaa`)

Arrays in Java

- Special objects with publicly visible (immutable) field `length`
- Fixed length: once constructed, the length cannot change
- Example: `int[] myNumbers = new int[10];`
- Object arrays do **not** construct objects, but merely an array of references (example: `String[] myNames = new String[3];`)
- `String[] myNames = new String[] {"tommi", "alex", "fred"};`
- Multidimensional arrays do not exist within computers (example: 2-dim int array)

Decisions: if/then/else

```
int myAge = 24;

if (myAge < 18) {
  System.out.println("Too young to study");
} else if (myAge >= 25) {
  System.out.println("Still studying huh?");
} else {
  System.out.println("Proud econometrics student");
}
```

Decisions: numerical comparison operators

```
5.0 < 10.0; // true
5 <= 5; // true
5.0 > 3.0; // true
5 >= 5; // true
5 == 10; // false
5 != 10; // true
```

Decisions: boolean operators

```
boolean a = false;
boolean b = true;

a | b; // OR: true
a & b; // AND: false
!a; // NOT: true
a ^ b; // XOR: true
a == b; // EQUAL TO: false
a != b; // NOT EQUAL TO: true
a || b; // shortcut OR
a && b; // shortcut AND

if (x != null && x.getValue() == 5) { ... }
```

Decisions: switch

```
int x = 2;

switch(x) {
  case 1:
  System.out.println("x is 1");
  break;

  case 2:
  System.out.println("x is 2");

  default:
  System.out.println("x is at least 2");
  break;
}
```

Iteration: for

```
double[] nrs = new double[] {1.0, 3.0, 5.5};

for (int i=0;i<nrs.length;i++) {
  System.out.println(i+"'th number is " + nrs[i]);
}
```

Iteration: enhanced for

```
double[] nrs = new double[] {1.0, 3.0, 5.5};

for (double nr : nrs) {
  System.out.println(”number is ” + nr);
}
```

Iteration: while

```
double[] nrs = new double[] {1.0, 3.0, 5.5};

int current = 0;

while (current < nrs.length) {
  System.out.println(current
  + ”’th number is ” + nrs[current]);

  current++;
}
```

Iteration: do/while

```
double[] nrs = new double[] {1.0, 3.0, 5.5};

int next = 0;

do {
  System.out.println(next
  + "'th number is " + nrs[next]);
  next++;
} while (next < nrs.length);
```

Errors and exceptions

- Method calls can fail due to
  - unexpected reasons mostly uncontrollable by the programmer: hard disk breaks down while reading a file, out of memory while requesting allocation via `new`
  - invalid input (next lecture)
- How to signal failure?

Return value outside the valid set

```
public int lengthOfString(String str) { ... }

public int parseInt(String str) { ... }
```

Exceptions

```
public int parseInt(String str)
  throws NumberFormatException { ... }
...
try {
  int myint = parseInt("1.0");
  myint = parseInt("5.5");
} catch (NumberFormatException e) {
  System.out.println("Cannot parse: "
  + e.getMessage());
}
```

Checked exceptions

- **Checked** exceptions are ones that the program has to be able to recover from, and therefore they have to be caught
- **Unchecked** exceptions are serious errors (out of memory) or logic errors (method call with invalid input values) that the program should not recover from
- Checked exceptions redirect control flow and their use should be minimized
- Unchecked exceptions in Java are in the type hierarchy below RuntimeException or Error

Errors

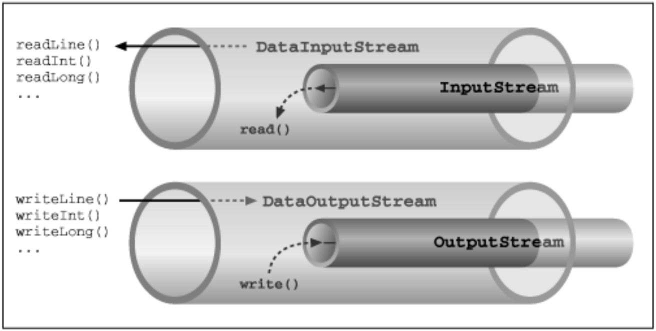
```
public int add(int a, int b) {
  int c = a;
  c += b;
  if (c != (a + b)) {
    throw new RuntimeException(”broken VM?”);
  }
}
```

I/O: streams

- Streams are abstraction of input/output in OOP
- Enable to transparently change transmission/storage media: reading from disk vs reading from web server
- `System.out` is a `PrintStream`
- Character sources/sinks can be read/written with Readers/Writers

Stream combination

- Lower level streams can be used by higher level streams to provide additional functionality
- Streams can perform input conversion transparently on the fly



```java
BufferedReader rdr = null;
try {
  rdr = new BufferedReader(
  new FileReader("file.txt")));
  String s = null;
  do {
    s = rdr.readLine();
    if (s!=null){System.out.println("Read:" + s);}
  } while (s != null);
} catch (FileNotFoundException e) {
  System.out.println("file.txt not found");
} catch (IOException e) {
  System.out.println("Error reading file: "
    + e.getMessage());
} finally {
  try {
    if (rdr != null) {
      rdr.close();
    } } catch (IOException e) { }
}
```

Premature optimization is the root of all evil

D.E. Knuth