CSSE2002/7023

Semester 2, 2021

Programming in the Large

Week 6.1: Miscellaneous Java

Reminders

- Assignment 1 due Sep 10 at 4pm
  - Do not leave submission until the last second
    - 16:00:01 is late
    - Committing and uploading files takes time – starting to upload at 15:59 is likely to result in a late submission
  - Remember that files need to be in the correct directories and have the correct names
    - Take advantage of Gradescope pre-check. If your assignment does not conform in a pre-check, it will not pass automated testing.
- To give you extra time and support for the assignment for Week 6 only:
  - Practical sessions will be dedicated to providing assignment help

In this Session

- `instanceof`
- Newlines
- Pre/Post Increment/Decrement
- Ternary Operator
- `final`

instanceof

expression `instanceof` *type*

Returns true if the value of the expression can be treated (with appropriate casts) as *type*.

- e.g.

```
("Hello" instanceof String) == true
```

- It is not the same as asking "Is this instance of precisely this class (i.e. not a subclass)?"

```
("Hello" instanceof Object) == true
```

- Does not tell us exactly what class the object was created as[1].

[1] .getClass() for that

Questionable Uses

It might be tempting to write code like:

```
... doStuff(Student s) {
    if (s instanceof UnderGrad) {
      // UnderGrad things
    } else if (s instanceof PostGrad) {
      // PostGrad things
    } else if (s instanceof Research) {
      // Research things
    }
}
```

Code like this could be spread all over the place

- Hard to maintain!

Remember *Encapsulation?*

Code that is specific to a type should be encapsulated in its class

- Why isn't "UnderGrad things" in the UnderGrad class?
- ... "PostGrad things" in the PosttGrad class?

Surely the following is better?

```
... doStuff(Student s) {
    s.doStuffHelper();
}
```

This way, if a new type of student needs to be added, we don't need to find all the places and correctly add all the new cases.

But what if I can't just add methods to the classes?[2]

[2] Custom wrapper classes maybe?

Newlines

Different operating systems have different ideas about how to encode a new line in characters:

- Unix type systems use - `'\n'`
- Windows uses - `'\r\n'`
- Old Macintosh(?) - `'\r'`

If you are writing code which will need to run on different systems (something you should consider), what do you do?

- println() will automatically use the correct one for your system.
- `System.lineSeparator()` or `String.format("%n")`[3] will give you the string if you need it.

[3] Read java.util.Formatter JavaDoc carefully, before trying anything complex with this one.

Pre/Post Increment/Decrement

```
x++; ++x; --y; y--;
```

Two factors:

- Affect they have on variables
  - `++` adds 1 to the variable
  - `--` subtracts 1
- What they evaluate to (suppose `x == 1`):

`int res = (++x)`	`x += 1;` `res = x;`	`x == 2, res == 2`
`int res = (x++)`	`res = x;` `x += 1;`	`x == 2, res == 1`
`int res = (--x)`	`x -= 1;` `res = x;`	`x == 0, res == 0`
`int res = (x--)`	`res = x;` `x -= 1;`	`x == 0, res == 1`

Ternary Operator

test_expression **?** *true_expression* **:** *false_expression*

- . . . is an expression
- if *test_expression* is true, has the value of *true_expression*
- else *false_expression*

```
int a = (x < 0) ? -x : x;
```

Same logic as:

```
int a;
if (x < 0) {
    a = -x;
} else {
    a = x;
}
```

`final` — Variables

The `final` keyword has two possible meanings depending on its context. For variables, the value cannot be modified after its first assignment:

```
public void stuff() {
    final int x = 5;
    x = 4; // error
}
```

Member variables (if not declared with a value), **must** be set *once* in each constructor, but nowhere else.

Symbolic Constants

```
public class Chem {
   public static final double AVAGADRO = 6.023e23;
   ...
}
```

Everyone can access `Chem.AVAGADRO`

- but no one can change it

`final` — Object Variables

`final` on a variable means that its value (i.e. its *reference*) can't be changed. It does not mean the object referred to can't change state.

```
public class Test {
  public int q;
  public Test(int v) {
      q = v;
  }

  public static void main(String args[]) {
      final Test x = new Test(10);
      x.q = 15;
  }
}
```

`final` — Methods

A method that is declared final cannot be overridden in sub-classes.

```
public class A {
    public final int getLBound() {
        return 0;
    }

    public int getUBound() {
        return 10;
    }
}
```

```
public class B extends A {
    @Override
    public int getLBound() {   // NO!
        return 1;
    }

    @Override
    public int getUBound() {   // OK
        return 9;
    }
}
```

final classes

```
public final class TheEnd {...}
```

- Member variables can't be changed once assigned.
- No subclass can extend the class.