Designing Objects, UML, and Communication

Overview

Designing Objects

- Instance variables
- Instance methods
- Constructors
- Public versus private
- `this`
- `toString`
- `equals`
- Eclipse will do the tedious work for you
- Separating model from view

Unified Modeling Language (UML) class diagrams

Communication

Designing Objects

Instance variables

Variables defined outside of any method:

```
public class Point {
   private double x;
   private double y;
}
```

Visible from any method.

Get default values (`0`, `false`, `null`) if not initialized.

Each instance gets its own copy.

Given an instance `p`, you can access `p.x`.

Instance methods

Not static.

```
public double getDistanceFromOrigin() {
   return Math.sqrt(x * x + y * y);
}
```

Called on a particular instance:

```
p.getDistanceFromOrigin()
```

Constructors

Same name as class, no return type.

```
public Point() {
    x = 0.0;
    y = 0.0;
}
```

Creating an object with `new` calls a constructor.

Instance variables should be initialized in constructors.

If you don’t define a constructor, you get a free zero-argument constructor that doesn’t do anything.

Public versus private

`private`: can only be seen in this class.

`public`: can be seen in any class.

Instance variables should be declared private. Methods can be public. Let others know what they can ask an object to do, not how it works.

Other classes should have to go through your methods to get at your instance variables:

`x++;` // From the same class

`p.setX(p.getX() + 1);` // From another class

This enforces encapsulation, making debugging easier.

(There are two other intermediate visibility levels, but they're only relevant in programs with multiple packages.)

`this`

Refers to the object on which a method is called.

Useful when an instance variable and an argument have the same name:

```
public Point(double x, double y) {
    this.x = x;
    this.y = y;
}
```

`this` is the extra “secret argument” passed to an instance method:

`distanceBetween(p, q);` // Procedural style

`p.distanceTo(q);` // Object oriented style

`toString`

```
@Override
public String toString() {
    return "(" + x + ", " + y + ")";
}
```

Returns a String representation of `this`.

Must take no arguments and return a String.

Handy for debugging and testing: what does my object look like?

Why `toString` and not `print`?

`@Override` is not strictly necessary, but can prevent mistakes.

`equals`

Compare objects with `a.equals(b)`, not `a == b`.

Unless you override the default, `equals` acts just like ==.

```
@Override
public boolean equals(Object obj) {
    if (this == obj) { return true; }
    if (obj == null) { return false; }
    if (getClass() != obj.getClass()) { return false; }
    Point other = (Point) obj;
    if (x != other.x) {
        return false;
    }
    if (y != other.y) {
        return false;
    }
    return true;
}
```

Note that argument type must be Object.

Eclipse will do the tedious work for you

Look in the Source menu.

Separating model from view

Put the logical model and the GUI in different classes.

It’s much easier to write automated tests for the logical model.

If you want to change (or replace) the UI, it’s all in one place.

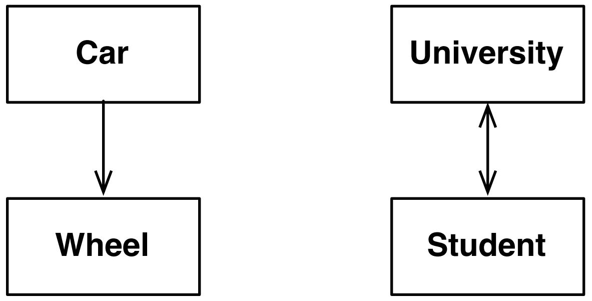
Keep similar things together.

“[Divide] things … by classes, where the natural joints are, and [do not try] to break any part, after the manner of a bad carver.” -Plato, *Phaedrus*

Unified Modeling Language (UML) class diagrams

A Car has-a Wheel (presumably several of them).

A University has-a Student, and vice versa.



Some UML class diagrams add other details, like listing the instance variable and method names.

There are other kinds of UML diagrams for other purposes.

Communication

Your results are nearly useless if you can’t communicate them.

Teams depend on communication to coordinate and avoid wasting effort.

Be clear and precise.

Use diagrams and analogies to express complex ideas.

“Omit needless words.” -Strunk & White, *The Elements of Style*

Listen actively (make eye contact, paraphrase).

Just because you said/wrote it doesn’t mean they got it.

Review

Classes contain instance variables, methods, and constructors.

Methods should be public, instance variables private.

`this` is the object on which a method was called.

Consider defining `toString` and `equals` (or making Eclipse do it).

Separate the model from the view.

UML class diagrams show the relationships between classes.

Strive to communicate clearly.