Objects and Classes

CS 99 – Summer 2000
Michael Clarkson
Lecture 8

Administration

- Review session 7:30pm, Phillips 203
- Prelim tomorrow
  - 10:00 in Upson 215
  - 11:30 in Upson B17
- Wednesday
  - Lab 8 due
  - Lecture 9
- Lab 9 on Thursday

Agenda

- Review of objects and classes
- In-class exercise

Writing Classes

- Until the last lab, we were only using predefined classes.
- Starting with that lab, we wrote our own class to define new objects

Objects

- An object has:
  - State: descriptive characteristics
  - Behavior: what it can do (or can be done to it)
- For example, consider a coin that can be flipped so that its face shows either "heads" or "tails"
- The state of the coin is its current face (heads or tails)
- The behavior of the coin is that it can be flipped
- Note that the behavior of the coin might change its state

Classes

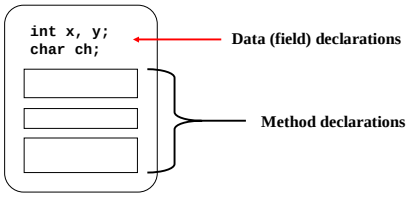
- A *class* is a blueprint of an object
- It is the model or pattern from which objects are created
- For example, the `String` class is used to define `String` objects
- Each `String` object contains specific characters (its state)
- Each `String` object can perform services (behaviors) such as `toUpperCase`

Classes [2]

- The `String` class was provided for us by the Java standard class library
- But we can also write our own classes that define specific objects that we need
- For example, suppose we wanted to write a program that simulates the flipping of a coin
- We could write a `Coin` class to represent a coin object

Classes [3]

- A class contains data and method declarations:

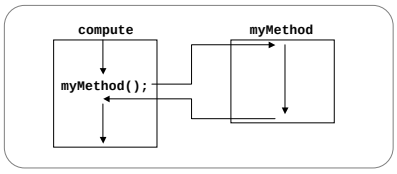


Data Scope

- Instance data declared at the class level can be used by all non-static methods in that class
- Static data declared at the class level can be used by all methods in that class
- Data declared within a method can only be used in that method
- Data declared within a method is called *local data*

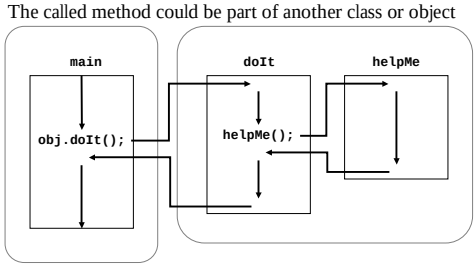
Calling Methods

- The called method could be within the same class, in which case only the method name is needed



Calling Methods [2]

- The called method could be part of another class or object



The `Coin` Class

- In our `Coin` class we could define the following fields:
  - `face`, an integer that represents the current face
  - `HEADS` and `TAILS`, integer constants that represent the two possible states
- We might also define the following methods:
  - a `Coin` constructor, to set up the object
  - a `flip` method, to flip the coin
  - a `getFace` method, to return the current face
  - a `toString` method, to return a string description for printing

The Coin Class [2]

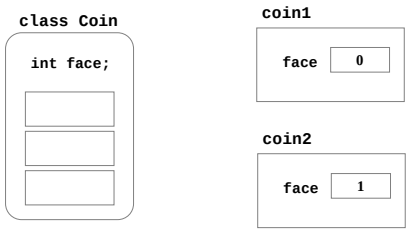
- Once the `Coin` class has been defined, we can use it again in other programs as needed
- A program will not necessarily use every service provided by an object

Instance Data

- The `face` variable in the `Coin` class is called *instance data* because each instance (object) of the `Coin` class has its own
- A class declares the type of the data, but it does not reserve any memory space for it
- Every time a `Coin` object is created, a new `face` variable is created as well
- The objects of a class share the method definitions, but they have unique data space
- That's the only way two objects can have different states

Instance Data [2]

```
Coin coin1 = new Coin();
Coin coin2 = new Coin();
```



Instance Data and Methods

- When a method is invoked on an object, or inside an object, that object's fields are in scope
- So given:

```
coin1.flip();
```

  - Control is transferred to `flip()`
  - `coin1`'s fields are in scope

this

- Inside a method, the object that the method was invoked on can be referred to with the special variable `this`
- So instead of writing:

```
face = ...
```

  we could also write

```
this.face = ...
```

- And instead of:

```
flip();
```

  we could also write:

```
this.flip();
```


Constructors

- A constructor is a special method that is used to set up a newly created object
- When Java encounters an object creation:

```
new Coin()
```

  It reserves memory for the object, then calls a constructor
- Remember that constructors:
  - Have the same name as the class
  - Do not return a value
  - Have no return type, not even `void`
  - Often set the initial values of instance variables