Messages vs. Methods

A message is a name (along with parameters):
- `object.drink(...)`

The method is the code that actually executes
- `public class Animal {
    public void drink() {
        ...
    }
}
- `public class Dog extends Animal {
    public void drink() {
        ...
    }
}

Technically, we don’t call an instance method in Java, we send a message.

Late binding

Polymorphism
- Meaning of the roots
- Meaning in Java:
The determination of which method to call is delayed until runtime
  - Must check the runtime-type of the object
The static-type determines the possible messages that can be sent
  - The run-time type determines which method will actually be executed

Late binding is only done for instance methods
Class methods use early binding

What-to-do versus how-to-do-it

A message is a request to carry out a task
- What to do

A method actually carries it out
- How to do it
  - Could be different for different subclasses
Examples

class Dog {
  public void bark() { make normal bark }
}
class Chihuahua extends Dog {
  public void bark() { make yip }
}
class Wolfhound extends Dog {
  public void bark() { make booming bark }
}

Given:
  - Dog aDog;
  ...
  aDog.bark();

which method executes?

How polymorphism works

Object has stored within it:
  - Instance variables
  - Run-time type

Static type controls:
  - What instance variables can be read/written
  - What messages can be sent

Runtime type controls:
  - What methods actually execute

Using polymorphism

class DogArray {
  private Dog[] dogs = new Dog[MAX_DOGS];
  private int numDogs = 0;

  public void addDog(Dog d) {dogs[numDogs++] = d;}
  public void makeAllDogsBark() {
    for (int i = 0; i < numDogs; i++)
      dogs[i].bark();
  }
}

DogArray arr = new DogArray();
arr.addDog(new Chihuahua());
arr.addDog(new Wolfhound());
arr.addDog(new Dog());
arr.makeAllDogsBark();

The clone method

How can you duplicate an object?
  - Copy constructor:
    - public Dog(Dog d) { ... }
  - Can lose the runtime type
    - Dog d = new Chihuahua();
    - Dog duplicateDog = new Dog(d);

We want to maintain the runtime type
  - Solution: clone
    - Dog d = new Chihuahua();
    - Dog duplicateDog = d.clone();

The Object class defines this method:
  - protected Object clone()
    - Intention of method:
      - Create a new object which is a duplicate of the original, but which
        shares no references (deep copy)
      - Object.clone() returns an exact duplicate of the object with the
        same values for instance variables (shallow copy)
Implementing Clone

class Dog {
  private dogInstanceVar;
  public void bark() { make normal bark }
  public Dog(Dog d) { dogInstanceVar = d.dogInstanceVar; }
  public Dog clone() { return new Dog(this); }
}
class Chihuahua extends Dog {
  private chihuahuaInstanceVar;
  public void bark() { make yip }
  public Chihuahua(Chihuahua c) {
    super(c);
    chihuahuaInstanceVar = c.chihuahuaInstanceVar;
  }
  public Chihuahua clone() { return new Chihuahua(this); }
}
Dog aDog = new Chihuahua(); Dog aNewDog = aDog.clone();

Actually more complicated
We'll revisit when we learn
about interfaces

Using clone

class DogArray {
  private Dog[] dogs = new Dog[MAX_DOGS];
  private int numDogs = 0;
  public void addDog(Dog d) { dogs[numDogs++] = d; }
  public void makeAllDogsBark() { … }
  public DogArray(DogArray a) {
    numDogs = a.numDogs;
    for (int i = 0; i < numDogs; i++)
      dogs[i] = a.dogs[i].clone();
  }
}
DogArray arr = new DogArray();
arr.addDog(new Chihuahua());
arr.addDog(new Wolfhound());
arr.addDog(new Dog());
DogArray newArr = DogArray(d);

Abstract classes and methods

An abstract method is one which is declared but has no default implementation

- What should the run method of Animal do? Every animal can run, but each one runs differently
  - No default is possible
- Solution: declare the method as abstract
  - public class Animal {
    …
    public abstract run();
  }
- An abstract method has no actual implementation
  - Subclasses will override the method, providing a concrete method

An abstract class is any class which has an abstract method

- Or, a class which has inherited an abstract method and not overridden it
- Abstract classes cannot be instantiated
  - new Animal() not allowed

Final methods and classes

Sometimes, you may not want a subclass to override a method

- You may want the speed of early binding
- You may not trust your subclass to do the right thing
- Solution:
  - public class Chihuahua extends Dog {
    public final void climb() { … }
  }

Sometimes, you may not want a class to be subclassed at all

- You may not trust your subclass
- You may see no need (e.g., Math, System all have only static methods)
- Solution:
  - public final class Math {
    …
  }
| Debugging demo |  |