CS2102: Lecture on Abstract Classes and Inheritance

Kathi Fisler
How to Use These Slides

These slides walk you through how to share common code (i.e., create helper methods) across classes

• I recommend you download the starter file (posted to the website) and make the edits in the slides, step by step, to see what happens for yourself

• In the slides, green highlights what changed in the code from the previous slide; yellow highlights show Java compile errors

• Note any questions, and ask on the board or in the lecture-time chat
interface IAnimal {
   // determine whether animal's length
   // is within normal boundaries
   boolean isNormalSize();
}

class Dillo implements IAnimal {
   int length;
   boolean isDead;

   Dillo(int length, boolean isDead) {
      this.length = length;
      this.isDead = isDead;
   }

   // determine whether this dillo's
   // length is between 2 and 3
   public boolean isNormalSize () {
      return 2 <= this.length &&
              this.length <= 3 ;
   }
}

class Boa implements IAnimal {
   int length;
   String eats;

   Boa(int length, String eats) {
      this.length = length;
      this.eats = eats;
   }

   // determine whether this boa's
   // length is between 5 and 10
   public boolean isNormalSize () {
      return 5 <= this.length &&
              this.length <= 10 ;
   }
}

Notice the almost identical code.
interface IAnimal {
    // determine whether animal's length
    // is within normal boundaries
    boolean isNormalSize();
}

class Dillo implements IAnimal {
    int length;
    boolean isDead;

    Dillo(int length, boolean isDead) {
        this.length = length;
        this.isDead = isDead;
    }

    // determine whether this dillo's
    // length is between 2 and 3
    public boolean isNormalSize () {
        return 2 <= this.length &&
                this.length <= 3 ;
    }
}

class Boa implements IAnimal {
    int length;
    String eats;

    Boa(int length, String eats) {
        this.length = length;
        this.eats = eats;
    }

    // determine whether this boa's
    // length is between 5 and 10
    public boolean isNormalSize () {
        return 5 <= this.length &&
                this.length <= 10 ;
    }
}

We should create a helper method, but where can we put it? (remember, all
methods must be in a class)

Notice the almost identical code
We will create a new class that abstracts over the common features of Dillo and Boa. We’ll call the new class AbsAnimal ("abs" for abstract).

```java
class Dillo implements IAnimal {
    int length;
    boolean isDead;
    Dillo(int length, boolean isDead) {
        this.length = length;
        this.isDead = isDead;
    }
    // determine whether this dillo's length is between 2 and 3
    public boolean isNormalSize() {
        return 2 <= this.length && this.length <= 3;
    }
}
class Boa implements IAnimal {
    int length;
    String eats;
    Boa(int length, String eats) {
        this.length = length;
        this.eats = eats;
    }
    // determine whether this boa's length is between 5 and 10
    public boolean isNormalSize() {
        return 5 <= this.length && this.length <= 10;
    }
}
```
We will put a helper method for `isNormalSize` in `AbsAnimal`.

We call the helper `isLenWithin`; it takes the varying low and high values as inputs (but otherwise copies the common code, as usual when making a helper).

```java
class AbsAnimal {
    // determine whether animal’s length is between low and high
    boolean isLenWithin (int low, int high) {
        return low <= this.length &&
                this.length <= high;
    }
}
```
Next, we rewrite the original `isNormalSize` methods to call the helper method

```java
interface IAnimal {
    // determine whether animal's length is within normal boundaries
    boolean isNormalSize();
}

class AbsAnimal {
    // determine whether animal's length is between low and high
    boolean isLenWithin(int low, int high) {
        return low <= this.length &&
                this.length <= high ;
    }
}

class Dillo implements IAnimal {
    int length;
    boolean isDead;

    Dillo(int length, boolean isDead) {
        this.length = length;
        this.isDead = isDead;
    }

    // determine whether this dillo's length is between 2 and 3
    public boolean isNormalSize() {
        return isLenWithin(2,3);
    }
}

class Boa implements IAnimal {
    int length;
    String eats;

    Boa(int length, String eats) {
        this.length = length;
        this.eats = eats;
    }

    // determine whether this boa's length is between 5 and 10
    public boolean isNormalSize() {
        return isLenWithin(5,10);
    }
}
```
This is the right idea, but if we compile the `Dillo` and `Boa` classes, Java will complain that `isLenWithin` isn’t defined.

```java
class AbsAnimal {
    // determine whether animal’s length is within normal boundaries
    boolean isLenWithin(int low, int high) {
        return low <= this.length && this.length <= high;
    }
}

class Dillo implements IAnimal {
    int length;
    boolean isDead;

    Dillo(int length, boolean isDead) {
        this.length = length;
        this.isDead = isDead;
    }

    // determine whether this dillo’s length is between 2 and 3
    public boolean isNormalSize() {
        return isLenWithin(2, 3);
    }
}

class Boa implements IAnimal {
    int length;
    String eats;

    Boa(int length, String eats) {
        this.length = length;
        this.eats = eats;
    }

    // determine whether this boa’s length is between 5 and 10
    public boolean isNormalSize() {
        return isLenWithin(5, 10);
    }
}
```
This is the right idea, but if we compile the Dillo and Boa classes, Java will complain that `isLenWithin` isn't defined.

The problem is that we never connected Dillo and Boa to AbsAnimal.

```java
class AbsAnimal {
    // determine whether animal’s
    // length is between low and high
    boolean isLenWithin(int low, int high) {
        return low <= this.length &&
                this.length <= high ;
    }
}

class Dillo implements IAnimal {
    int length;
    boolean isDead;

    Dillo(int length, boolean isDead) {
        this.length = length;
        this.isDead = isDead;
    }

    // determine whether this dillo's
    // length is between 2 and 3
    public boolean isNormalSize () {
        return isLenWithin(2,3);
    }
}

class Boa implements IAnimal {
    int length;
    String eats;

    Boa(int length, String eats) {
        this.length = length;
        this.eats = eats;
    }

    // determine whether this boa's
    // length is between 5 and 10
    public boolean isNormalSize () {
        return isLenWithin(5,10);
    }
}
```

We connect Dillo and Boa to AbsAnimal using a new Java keyword, extends, which says that one class (Dillo/Boa) includes the content of another (AbsAnimal)

class AbsAnimal {
   // determine whether animal’s
   //  length is between low and high
   boolean isLenWithin (int low,
                       int high) {
       return low <= this.length &&
              this.length <= high ;
   }
}

class Dillo extends AbsAnimal implements IAnimal {
   int length;
   boolean isDead;

   Dillo(int length, boolean isDead) {
       this.length = length;
       this.isDead = isDead;
   }

   // determine whether this dillo's
   //    length is between 2 and 3
   public boolean isNormalSize () {
       return isLenWithin(2,3);
   }
}

class Boa extends AbsAnimal implements IAnimal {
   int length;
   String eats;

   Boa(int length, String eats) {
       this.length = length;
       this.eats = eats;
   }

   // determine whether this boa's
   //    length is between 5 and 10
   public boolean isNormalSize () {
       return isLenWithin(5,10);
   }
}
Now, AbsAnimal won’t compile; Java will say that it doesn’t have a length variable.

interface IAnimal {
    // determine whether animal's length is within normal boundaries
    boolean isNormalSize();
}

class AbsAnimal {
    // determine whether animal’s length is between low and high
    boolean isLenWithin (int low, int high) {
        return low <= this.length &&
                this.length <= high;
    }
}

class Dillo extends AbsAnimal implements IAnimal {
    int length;
    boolean isDead;

    Dillo(int length, boolean isDead) {
        this.length = length;
        this.isDead = isDead;
    }

    // determine whether this dillo's length is between 2 and 3
    public boolean isNormalSize () {
        return isLenWithin(2,3);
    }
}

class Boa extends AbsAnimal implements IAnimal {
    int length;
    String eats;

    Boa(int length, String eats) {
        this.length = length;
        this.eats = eats;
    }

    // determine whether this boa's length is between 5 and 10
    public boolean isNormalSize () {
        return isLenWithin(5,10);
    }
}
Now, AbsAnimal won’t compile; Java will say that it doesn’t have a length variable.

But note that the length variable is also common to Dillo and Boa. It should also have moved to AbsAnimal.

```java
class AbsAnimal {
    // determine whether animal’s length is between low and high
    boolean isLenWithin(int low, int high) {
        return low <= this.length && this.length <= high;
    }
}

class Dillo extends AbsAnimal implements IAnimal {
    int length;
    boolean isDead;

    Dillo(int length, boolean isDead) {
        this.length = length;
        this.isDead = isDead;
    }

    // determine whether this dillo's length is between 2 and 3
    public boolean isNormalSize() {
        return isLenWithin(2, 3);
    }
}

class Boa extends AbsAnimal implements IAnimal {
    int length;
    String eats;

    Boa(int length, String eats) {
        this.length = length;
        this.eats = eats;
    }

    // determine whether this boa's length is between 5 and 10
    public boolean isNormalSize() {
        return isLenWithin(5, 10);
    }
}
```
Now, `AbsAnimal` won’t compile; Java will say that it doesn’t have a length variable.

But note that the `length` variable is also common to `Dillo` and `Boa`. It should also have moved to `AbsAnimal`.
We need to add a constructor to `AbsAnimal`, and have it set the value of `length`.

[For sake of space, we will hide the `Boa` class (edits to `Dillo` apply to `Boa` as well)]

```java
class AbsAnimal {
    int length;

    // constructor
    AbsAnimal(int length) {
        this.length = length;
    }

    // determine whether animal's length is between low and high
    boolean isLenWithin(int low, int high) {
        return low <= this.length && this.length <= high;
    }
}
```

```java
class Dillo extends AbsAnimal implements IAnimal {
    boolean isDead;

    Dillo(int length, boolean isDead) {
        this.length = length;
        this.isDead = isDead;
    }

    // determine whether this dillo's length is between 2 and 3
    public boolean isNormalSize() {
        return isLenWithin(2, 3);
    }
}
```

Notice that we removed the `length` variable from `Dillo`.

We need to add a constructor to `AbsAnimal`, and have it set the value of `length`.

interface IAnimal {
    // determine whether animal's length
    // is within normal boundaries
    boolean isNormalSize();
}

class Dillo extends AbsAnimal implements IAnimal {
    boolean isDead;

    Dillo(int length, boolean isDead) {
        super(length);
        this.isDead = isDead;
    }

    // determine whether this dillo's
    // length is between 2 and 3
    public boolean isNormalSize() {
        return isLenWithin(2,3);
    }
}

class AbsAnimal {
    int length;

    // constructor
    AbsAnimal(int length) {
        this.length = length;
    }

    // determine whether animal’s
    // length is between low and high
    boolean isLenWithin (int low, int high) {
        return low <= this.length &&
               this.length <= high;
    }
}
In Java, `super` refers to the constructor for the class that this class extends; inside Dillo, `super` calls the AbsAnimal constructor.

In Java,

```java
class AbsAnimal {
    int length;
    // constructor
    AbsAnimal(int length) {
        this.length = length;
    }
    // determine whether animal’s length is between low and high
    boolean isLenWithin (int low, int high) {
        return low <= this.length && this.length <= high;
    }
}

interface IAnimal {
    // determine whether animal’s length is within normal boundaries
    boolean isNormalSize();
}
```

```java
class Dillo extends AbsAnimal implements IAnimal {
    boolean isDead;
    Dillo(int length, boolean isDead) {
        super(length);
        this.isDead = isDead;
    }
    // determine whether this dillo’s length is between 2 and 3
    public boolean isNormalSize () {
        return isLenWithin(2,3);
    }
}
```

Notice that we removed the length variable from Dillo.
The Dillo constructor needs to send the length value to the AbsAnimal constructor.
In Java, `super` refers to the constructor for the class that this class extends; inside `Dillo`, `super` calls the `AbsAnimal` constructor.

```java
class Dillo extends AbsAnimal implements IAnimal {
    boolean isDead;

    Dillo(int length, boolean isDead) {
        super(length);
        this.isDead = isDead;
    }

    // determine whether this dillo's length is between 2 and 3
    public boolean isNormalSize() {
        return isLenWithin(2, 3);
    }
}
```

Whenever a class extends another class, its constructor should call `super` before doing anything else (i.e., the call to `super` should be the first statement in the method)

```java
interface IAnimal {
    // determine whether animal's length is within normal boundaries
    boolean isNormalSize();
}
```

```java
class AbsAnimal {
    int length;

    // constructor
    AbsAnimal(int length) {
        this.length = length;
    }

    // determine whether animal's length is between low and high
    boolean isLenWithin (int low, int high) {
        return low <= this.length &&
                this.length <= high ;
    }
}
```
class AbsAnimal implements IAnimal {
    int length;

    // constructor
    AbsAnimal(int length) {
        this.length = length;
    }

    // determine whether animal's length is between low and high
    boolean isLenWithin(int low, int high) {
        return low <= this.length && this.length <= high;
    }
}

class Dillo extends AbsAnimal implements IAnimal {
    boolean isDead;

    Dillo(int length, boolean isDead) {
        super(length);
        this.isDead = isDead;
    }

    // determine whether this dillo's length is between 2 and 3
    public boolean isNormalSize() {
        return isLenWithin(2, 3);
    }
}

Almost done. Since Dillo and Boa both implement IAnimal, we can move that to AbsAnimal as well.

Since Dillo and Boa both implement IAnimal, we can move that to AbsAnimal as well.

class AbsAnimal implements IAnimal {
    int length;

    // constructor
    AbsAnimal(int length) {
        this.length = length;
    }

    // determine whether animal's length is between low and high
    boolean isLenWithin(int low, int high) {
        return low <= this.length && this.length <= high;
    }
}

Almost done. Since Dillo and Boa both implement IAnimal, we can move that to AbsAnimal as well.

Since Dillo and Boa both implement IAnimal, we can move that to AbsAnimal as well.

class AbsAnimal implements IAnimal {
    int length;

    // constructor
    AbsAnimal(int length) {
        this.length = length;
    }

    // determine whether animal's length is between low and high
    boolean isLenWithin(int low, int high) {
        return low <= this.length && this.length <= high;
    }
}
interface IAnimal {
    // determine whether animal's length
    // is within normal boundaries
    boolean isNormalSize();
}

class Dillo extends AbsAnimal {
    boolean isDead;

    Dillo(int length, boolean isDead) {
        super(length);
        this.isDead = isDead;
    }

    // determine whether this dillo's
    // length is between 2 and 3
    public boolean isNormalSize () {
        return isLenWithin(2,3);
    }
}

class Boa extends AbsAnimal {
    String eats;

    Boa(int length, String eats) {
        super(length);
        this.eats = eats;
    }

    // determine whether this boa's
    // length is between 5 and 10
    public boolean isNormalSize () {
        return isLenWithin(5,10);
    }
}
class AbsAnimal implements IAnimal {
    int length;

    // constructor
    AbsAnimal(int length) {
        this.length = length;
    }

    // determine whether animal’s
    // length is between low and high
    boolean isLenWithin (int low,
          int high) {
        return low <= this.length &&
                this.length <= high ;
    }
}
Recap so far

• When multiple classes need to share code (such as a helper method), put that code in a (parent) class that the sharing classes each extend.

• Common variables and `implements` statements also move to the parent class.

• If a class extends another class, its constructor should call `super` (to properly set up the contents of the superclass).

• Classes can use all variables and methods in their superclass.
Facts about `Extends`

• **Terminology:** If class A extends class B, then (1) B is the *superclass* of A; (2) A is a *subclass* of B; (3) A is also said to *inherit* from B

• **Restrictions:** A class may have at most one superclass (ie, only *extends* one class), but arbitrarily many subclasses. [In contrast, a class can *implement* arbitrarily many interfaces.]

• **Behavior:** A class has access to all variables and methods of its superclass (there are exceptions, but we will discuss those later)

• **Behavior:** A class cannot access the variables or methods of its subclasses
BUT THERE ARE STILL SOME ISSUES TO ADDRESS …
class AbsAnimal implements IAnimal {
    int length;

    // constructor
    AbsAnimal(int length) {
        this.length = length;
    }

    // determine whether animal's length is between low and high
    boolean isLenWithin(int low, int high) {
        return low <= this.length && this.length <= high;
    }
}

class Dillo extends AbsAnimal {
    boolean isDead;

    Dillo(int length, boolean isDead) {
        super(length);
        this.isDead = isDead;
    }

    // determine whether this dillo's length is between 2 and 3
    public boolean isNormalSize() {
        return isLenWithin(2, 3);
    }
}

class Boa extends AbsAnimal {
    String eats;

    Boa(int length, String eats) {
        super(length);
        this.eats = eats;
    }

    // determine whether this boa's length is between 5 and 10
    public boolean isNormalSize() {
        return isLenWithin(5, 10);
    }
}
What if someone writes `new AbsAnimal(8)`?

What kind of animal does this yield?

It doesn’t yield any known (or meaningful) kind of animal. **AbsAnimal** is only meant to hold code, it shouldn’t be used to create objects.

We’d like to tell Java not to let anyone create objects from **AbsAnimal**.
abstract class AbsAnimal implements IAnimal {
    int length;

    // constructor
    AbsAnimal(int length) {
        this.length = length;
    }

    // determine whether animal’s length is between low and high
    boolean isLenWithin (int low, int high) {
        return low <= this.length &&
                this.length <= high ;
    }
}

class Boa extends AbsAnimal {
    String eats;

    Boa(int length, String eats) {
        super(length);
        this.eats = eats;
    }

    // determine whether this boa’s length is between 5 and 10
    public boolean isNormalSize () {
        return isLenWithin(5,10);
    }
}

To tell Java not to let anyone create objects from a class, we annotate the class with the keyword abstract

Now, the expression new AbsAnimal(8) would raise a Java error

Rule of thumb: if a class only to hold common code, make it abstract

// determine whether this dillo's length is between 2 and 3
public boolean isNormalSize () {
    return isLenWithin(2,3);
}
WHY DO WE NEED BOTH AN INTERFACE AND AN ABSTRACT CLASS?
abstract class AbsAnimal implements IAnimal {
    int length;

    // constructor
    AbsAnimal(int length) {
        this.length = length;
    }

    // determine whether animal’s length is between low and high
    boolean isLenWithin (int low, int high) {
        return low <= this.length &&
        this.length <= high ;
    }
}

interface IAnimal {
    // determine whether animal's length is within normal boundaries
    boolean isNormalSize();
}

Interfaces and abstract classes serve two very different purposes

Interfaces are a form of types: they capture what a class must do, but they do not constrain how the class does something. As such, interfaces cannot contain code (beyond method input/output types) or variables.

Abstract classes are for sharing (abstracting over) data and code across multiple classes; they constrain how extending classes organize and use data.

Both roles are important, so OO programs often use both
interface IAnimal {
    // determine whether animal's length
    // is within normal boundaries
    boolean isNormalSize();
}

abstract class AbsAnimal implements IAnimal {
    int length;

    // constructor
    AbsAnimal(int length) {
        this.length = length;
    }

    // determine whether animal’s
    // length is between low and high
    boolean isLenWithin (int low, int high) {
        return low <= this.length &&
              this.length <= high;
    }
}

Interfaces and abstract classes serve two very different purposes

Imagine that we wanted to add fruit flies to our data. They are too small to have a length. Having IAnimal lets us write isNormalSize (to always return true) without having to specify a meaningless length value for a fruit fly.

If you already know some Java, you may have been taught to overuse class extension instead of interfaces. Interfaces are proper OO design practice (more on this through 2102)
What you should be able to do now ...

• Use `extends` to share code among classes

• Use `super` in constructors

• Make a class `abstract` to prevent someone from creating objects from it

• Choose between using interfaces and (abstract) classes when designing programs
Some Study Questions

• Why didn’t we put `isLenWithin` in `IAAnimal`?

• Can `AbsAnimal` refer to the `eats` variable of `Boa`?

• Could we have defined `isNormalSize` directly inside of `AbsAnimal`, instead of writing `isLenWithin`? If so, how?

• If we wanted to write a `doesEatTofu` method on `Boa`, which class should it go into? Should it be mentioned in `IAAnimal`?
Experiments to Try on the Code

Edit the posted starter file with the code from these notes, then experiment with the following:

• What error does Java give if you try to extend an interface or implement an abstract class?

• What error does Java give if you try to access a subclass variable in a superclass?

• If you forgot to delete the `int length` line from the Dillo class (after adding it to AbsAnimal), what would Java do?