Abstract classes

Inheritance

Inheritance is the notion of one class inheriting the fields and methods from another class.

- In Java, inheritance happens when one class is defined as a *subclass* of another.
- Inheritance allows us to organize classes in hierarchical structures.
- Inheritance prevents redundancy.

Abstract classes

- In biology, to categorize species effectively, we often introduce *abstract* species (for example, mammals).
- Java supports a mechanism for such abstraction as well.
- In Java, an *abstract class* is a class designed to group similar classes together, *without implementing methods*.

Abstract classes

- An abstract class has methods with no bodies. Such methods are called *abstract methods*.
- An abstract class all by itself is not useful; in particular, we *cannot* instantiate an object of an abstract class.
- An abstract class becomes useful when used with its subclasses.

Abstract class `Mammal`

```
public abstract class Mammal {

  protected char gender;

  public Mammal() { }
  public String toString() {
    return "Gender: " + gender;
  }
  public abstract void breathe();
}
```

The method breathe() is *abstract* and has no body (i.e., implementation).

Abstract class `Mammal`

```
public abstract class Mammal {

  protected char gender;

  public Mammal() { }
  public String toString() {
    return "Gender: " + gender;
  }
  public abstract void breathe();
}
```

This class all by itself is useless. It becomes useful only when used with its subclasses.

Dog as a subclass of Mammal

```
public class Dog extends Mammal {

  // Dog inherits everything from Mammal
  protected String breed;
    ...

  // This implements Mammal's abstract method breathe().
  public void breathe() {
    System.out.println("This dog is breathing.");
  }
}
```

In a subclass, we implement the abstract method `breathe()` defined in `Mammal`.

Cat as a subclass of Mammal

```
public class Cat extends Mammal {

  // Cat inherits everything from Mammal
  protected String coatPattern;
    ...

  // This implements Mammal's abstract method breathe().
  public void breathe() {
    System.out.println("This cat is breathing.");
  }
}
```

In a subclass, we implement the abstract method `breathe()` defined in `Mammal`.

Abstract classes

Let `A` be an abstract class and `B` be a subclass of `A`.

- `B` inherits both fields and regular methods defined in `A`.
- In addition, in `B`, we must implement all abstract methods defined in `A` as regular methods.
- `B` can also have its own methods not defined in `A`.

Accessibility

Classes, fields and methods must be declared with *access modifiers*.

- Classes and methods are usually declared as `public`—accessible by anyone from anywhere.
- Fields are usually declared as `private`—accessible only within the same class.

Accessibility

For super-classes (including abstract classes), `private` is too restrictive. In such classes,

- methods are usually declared as `public`—accessible by anyone from anywhere.
- fields are usually declared as `protected`—accessible within the same class *and its subclasses*.

References

Let `A` be a class and `B` be a subclass of `A`.

- A reference variable of type `A` may refer to an object of type either `A` or `B`.
- A reference variable of type `B` may refer to only an object of type `B`.

References

If `PoliceDog` is a subclass of a class `Dog`, then the following are both legal:

```
Dog lucy = new Dog();
Dog rusty = new PoliceDog();
```

The following, however, is not:

```
PoliceDog winston = new Dog();
```


References

If `Cat` is a subclass of an abstract class `Mammal`, then the following are both legal:

```
Cat abe = new Cat();
Mammal hudson = new Cat();
```

The following, however, is not:

```
Cat fancy = new Mammal();
```