Abstract Classes and Methods

- Abstract classes
  - Sometimes it's useful to declare classes for which you never intend to create objects.
  - Used only as superclasses in inheritance hierarchies, so they are sometimes called abstract superclasses.
  - Cannot be used to instantiate objects—abstract classes are incomplete.
  - Subclasses must declare the "missing pieces" to become "concrete" classes, from which you can instantiate objects; otherwise, these subclasses, too, will be abstract.
- An abstract class provides a superclass from which other classes can inherit and thus share a common design.

Abstract Classes and Methods

- Classes that can be used to instantiate objects are called concrete classes.
- Such classes provide implementations of every method they declare (some of the implementations can be inherited).
- Abstract superclasses are too general to create real objects—they specify only what is common among subclasses.
- Concrete classes provide the specifics that make it reasonable to instantiate objects.
- Not all hierarchies contain abstract classes.

Abstract Classes and Methods

- Programmers often write programs that uses only abstract superclass types to reduce code's dependencies on a range of subclass types.
  - You can write a method with a parameter of an abstract superclass type.
  - When called, such a method can receive an object of any concrete class that directly or indirectly extends the superclass specified as the parameter's type.
- Abstract classes sometimes constitute several levels of a hierarchy.

Abstract Classes and Methods

- You make a class abstract by declaring it with keyword abstract.
- An abstract class normally contains one or more abstract methods.
  - An abstract method is one with keyword abstract in its declaration, as in

```
abstract void sound(); // abstract method
```

- Abstract methods do not provide implementations.
- A class that contains abstract methods must be an abstract class even if that class contains some concrete (nonabstract) methods.
- Each concrete subclass of an abstract superclass also must provide concrete implementations of each of the superclass's abstract methods.
- Constructors and static methods cannot be declared abstract.

Abstract Classes and Methods

- Cannot instantiate objects of abstract superclasses, but you can use abstract superclasses to declare variables
  - These can hold references to objects of any concrete class derived from those abstract superclasses.
  - Programs typically use such variables to manipulate subclass objects polymorphically.
- Can use abstract superclass names to invoke `static` methods declared in those abstract superclasses.

Abstract Classes Sytax

- A class that you define represents an abstract concept and, as such, cannot be instantiated
- An abstract class is a class that can only be subclassed, not instantiated.
- Use Abstract classes when you want to make sure a method has to be implemented in a subclass.
- Declaration of an abstract class:

```
abstract class Animal{
   . . .
   }
```

Abstract Method Syntax

```
public abstract class Animal{
    private String name;
    Animal(String name) { this.name = name;}
    abstract void sound();
}
```

An abstract method must be in an abstract class.

Abstract method sound provides a method interface for class Animal only.

Pure Abstract Class

- An abstract class may contain *abstract methods*, that is, methods with no implementation.

```
abstract class Animal {
        abstract void sound();
        abstract void feed();
};
```

Subclass Extending Abstract Class

Each non-abstract subclass of Animal, such as Dog would have to provide an implementation for the sound and feed method.

```
class  Dog extends Animal{
    void sound(){}
    void feed(){}
};
```

Subclass Extending Abstract Class

- Not all subclasses of an abstract class must be complete. If a subclass does not implement all abstract methods from its super class, then it must be an abstract class.

```
abstract class Animal{
    private String name;
    Animal(String name){this.name = name;}
        abstract void sound();
    abstract void feed();
            }
```

Class Dog does define behavior for sound() and feed() so it is a class that could be instantiated.

Class Cat does not define behavior for feed() so it is still an abstract class.

```
class Dog extends Animal{
        void sound(){}
    void feed() {}
}
```

```
abstract class Cat extends Animal{
     void sound(){}
}
```

When to use abstract class and abstract methods

- Define abstract super class when:
  - You don’t want the super class to be instantiated. For example, in a program it may not make sense to have an Animal object because Animal represents an abstract thing.
- Define abstract methods when:
  - You want to enforce a contract or a behavior on the subclasses.