

# Theory: Abstract class

🕒 18 minutes

0 / 5 problems solved

Skip this topic

Start practicing

3892 users solved this topic. Latest completion was about 6 hours ago.

Sometimes you have a set of fields and methods that you need to reuse in all classes within a hierarchy. It is possible to put all the common members to a special base class and then declare subclasses which can access these members. At the same time, you do not need to create objects of the base class. To achieve it, you can use an **abstract class** as the base class in the hierarchy.

## §1. What is an abstract class?

An **abstract class** is a class declared with the keyword `abstract`. It represents an abstract concept that is used as a base class for subclasses.

Abstract classes have some special features:

- it's impossible to create an instance of an abstract class;
- an abstract class can contain abstract methods that must be implemented in non-abstract subclasses;
- it can contain fields and non-abstract methods (including static);
- an abstract class can extend another class, including abstract;
- it can contain a constructor.

As you can see, an abstract class has two main differences from regular (concrete) classes: **no instances** and **abstract methods**.

**Abstract methods** are declared by adding the keyword `abstract`. They have a declaration (modifiers, a return type, and a signature) but don't have an implementation. Each concrete (non-abstract) subclass must implement these methods.

Note, static methods can't be abstract!

## §2. Example

Here is an abstract class `Pet`:

```
1 public abstract class Pet {
2
3     protected String name;
4     protected int age;
5
6     protected Pet(String name, int age) {
7         this.name = name;
8         this.age = age;
9     }
10
11     public abstract void say(); // an abstract method
12 }
```

The class has two fields, a constructor, and an abstract method.

Since `Pet` is an abstract class we cannot create instances of this class:

```
1 Pet pet = new Pet("Unnamed", 5); // this throws a compile time error
```

The method `say()` is declared abstract because, at this level of abstraction, its implementation is unknown. Concrete subclasses of the class `Pet` should have an implementation of this method.

Current topic:

Abstract class ...

Topic depends on:

✗ Polymorphism ...

Topic is required for:

Abstract class vs interface ...

✓ Anonymous classes ...

Template method ...

Table of contents:

↑ Abstract class

| §1. What is an abstract class?

§2. Example

Feedback & Comments

Below are two concrete subclasses of `Pet`. You can see that they override the abstract method:

```
1 class Cat extends Pet {
2
3     // It can have additional fields as well
4
5     public Cat(String name, int age) {
6         super(name, age);
7     }
8
9     @Override
10    public void say() {
11
12        System.out.println("Meow!");
13    }
14 }
15
16 class Dog extends Pet {
17
18     // It can have additional fields as well
19
20    public Dog(String name, int age) {
21        super(name, age);
22    }
23
24    @Override
25    public void say() {
26
27        System.out.println("Woof!");
28    }
29 }
```

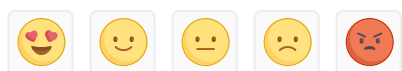
We can create instances of these classes and call the `say()` method.

```
1 Dog dog = new Dog("Boss", 5);
2 Cat cat = new Cat("Tiger", 2);
3
4 dog.say(); // it prints "Woof!"
5 cat.say(); // it prints "Meow!"
```

Do not forget that Java doesn't support multiple inheritance for classes. Therefore, a class can extend only one abstract class.

 Report a typo

**354** users liked this theory. **6** didn't like it. What about you?



Start practicing

[Comments \(19\)](#)

[Hints \(0\)](#)

[Useful links \(0\)](#)

[Show discussion](#)