Inner classes

Inner classes

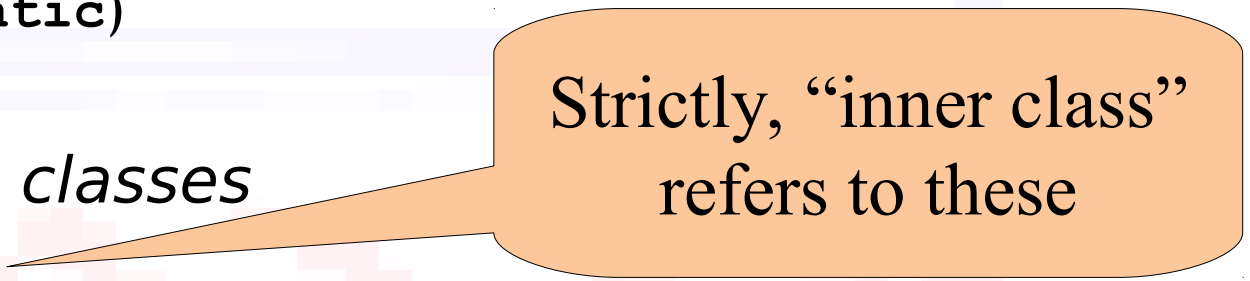
- A class can contain
  - Variables
    - instance
    - class (**`static`**)
  - Methods
    - instance
    - class (**`static`**)

Inner classes

- A class can contain
  - Variables
    - instance
    - class (**`static`**)
  - Methods
    - instance
    - class (**`static`**)
  - …but also *classes*
    - instance
    - class (**`static`**)

Inner classes

- A class can contain
  - Variables
    - instance
    - class (**`static`**)
  - Methods
    - instance
    - class (**`static`**)
  - …but also *classes*
    - instance
    - class (**`static`**)



Nested classes

- *Top-level* class
  - Contained in a package

```
package foo;
class Toto {
   ...
}

class Titi {
   ...
}
```

- *Nested* class
  - Contained in a class

```
package foo;
class Toto {
   ...
   class Titi {
      ...
   }
   ...
}
```

Nested classes

- Like variables and methods
- Same rights
  - Access to containing class
    - Can access its **`private`** members
    - Static nested classes can only access class members – no access to instance members

Inner classes

- Static nested classes
  - Belong to class **`Toto`**

```
class Toto {

   static class Titi {
   }

}
```

Inner classes

- Static nested classes
  - Belong to class **`Toto`**

```
class Toto {

   static class Titi {
   }

}

Toto.Titi tt
  = new Toto.Titi();
```

Inner classes

- Static nested classes
  - Belong to class **`Toto`**
- Inner classes
  - Belong to an object of type **`Toto`**

```
class Toto {

   static class Titi {
   }

}

Toto.Titi tt
  = new Toto.Titi();
```

Inner classes

- Static nested classes
  - Belong to class **Toto**

```
class Toto {

   static class Titi {
   }

}

Toto.Titi tt
  = new Toto.Titi();
```

- Inner classes
  - Belong to an object of type **Toto**

```
class Toto {

   class Titi {
   }

}
```

Inner classes

- Static nested classes
  - Belong to class **Toto**

```
class Toto {

   static class Titi {
   }

}

Toto.Titi tt
  = new Toto.Titi();
```

- Inner classes
  - Belong to an object of type **Toto**

```
class Toto {

   class Titi {
   }

}

Toto.Titi tt
  = new Toto().new Titi();
```

Anonymous inner classes

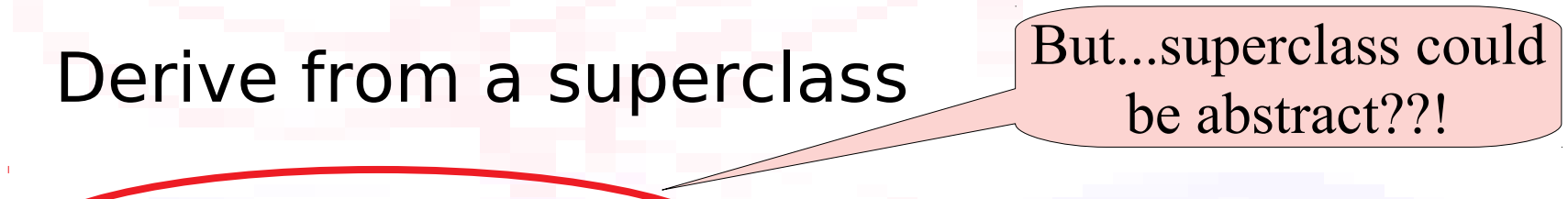
- Derive from a superclass

```
new superclass name([args list]) {
  anonymous class definition
}
```

- No name ⇒ no constructor
- Args passed to superclass constructor
- No interface ⇒ no **`implements`**

Anonymous inner classes

- Derive from a superclass



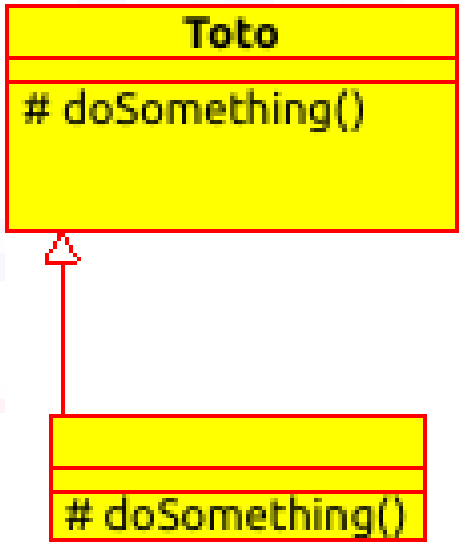

```
new superclass name([args list]) {
  anonymous class definition
}
```

- No name ⇒ no constructor
- Args passed to superclass constructor
- No interface ⇒ no **`implements`**

Anonymous inner classes

- ??? derives from superclass **`Toto`**

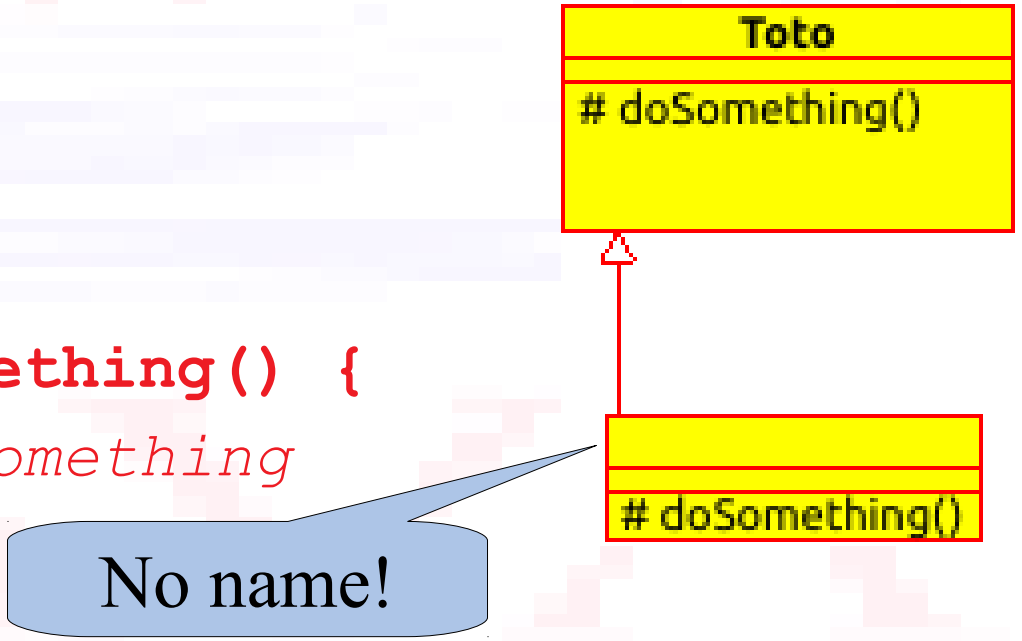
```
abstract class Toto {
    abstract void doSomething();
}

new Toto() {
    @Override
    void doSomething() {
        does something
    }
}
```



Anonymous inner classes

- ??? derives from superclass **`Toto`**

```
abstract class Toto {
    abstract void doSomething();
}

new Toto() {
    @Override
    void doSomething() {
        does something
    }
}
```



Anonymous inner classes

- Implement an interface

```
new interface name() {
  anonymous class definition
}
```

- Interface has no constructor ⇒ no args
- Derives from **`Object`** ⇒ no **`extends`**

Anonymous inner classes

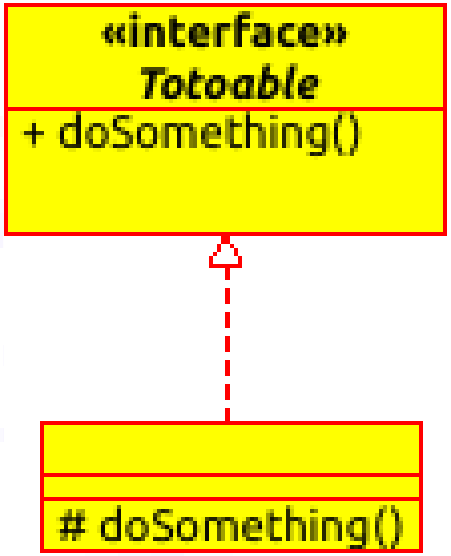
- Implement an interface


```
new interface name() {
  anonymous class definition
}
```

- Interface has no constructor ⇒ no args
- Derives from **`Object`** ⇒ no **`extends`**

Anonymous inner classes

- ??? implements an interface




```
interface Totoable {
    void doSomething();
}

new Totoable() {
    @Override
    public void doSomething() {
        does something
    }
}
```

Usage

- Wherever you need an object

```
class Toto {
   ...
   Titi aMethod(Tutu t) {
     Tata tata = new Tata() {…}
     return new Titi() {…}
     }
   }
}

toto.aMethod(new Tutu() {…});
```

Usage

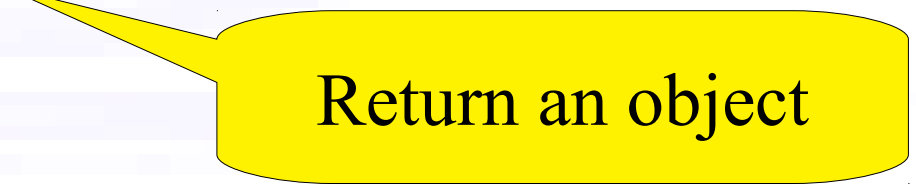
- Wherever you need an object

```
class Toto {
   ...
   Titi aMethod(Tutu t) {
     Tata tata = new Tata() {…}
     return new Titi() {…}
     }
   }
}

toto.aMethod(new Tutu() {…});
```

Usage

- Wherever you need an object

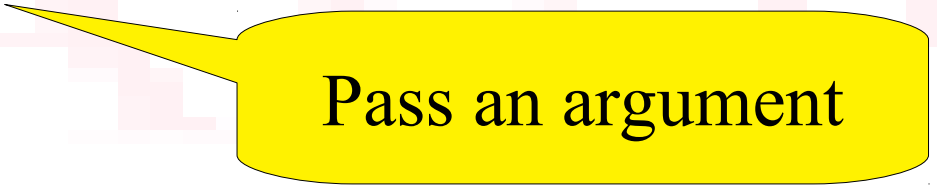
```
class Toto {
   ...
   Titi aMethod(Tutu t) {
     Tata tata = new Tata() {…}
     return new Titi() {…}
     }
   }
}

toto.aMethod(new Tutu() {…});
```



Usage

- Wherever you need an object

```
class Toto {
   ...
   Titi aMethod(Tutu t) {
     Tata tata = new Tata() {…}
     return new Titi() {…}
     }
   }
}

toto.aMethod(new Tutu() {…});
```



Usage

- An anonymous inner class is appropriate where:
  - definition is short
  - need a single instance
  - don’t need a constructor
  - object used immediately after creation
  - name doesn’t add to code clarity
- Use whenever it makes code clearer
  - Most often used in event-based code
- Allows testing abstract classes, interfaces