Relationships between objects II

Christian Rodríguez Bustos

Object Oriented Programming

Agenda

- Inheritance and Access Modifiers
- Overriding methods
- Reusing superclass behaviors

Inheritance and accessibility

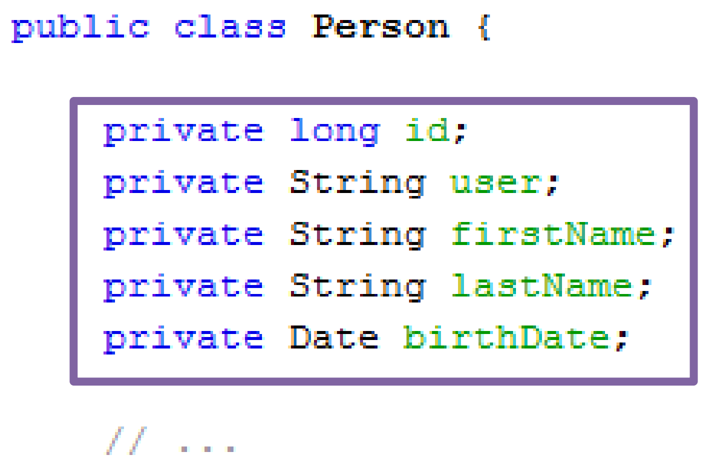
Java Access Modifiers
Inheritance and accessibility

Java Access Modifiers

Modifier	Access Levels			
	Class	Package	Subclass	World
`public`	☺	☺	☺	☺
`protected`	☺	☺	☺	☹
Default (no modifier)	☺	☺	☹	☹
`private`	☺	☹	☹	☹

Access level modifiers determine **whether other classes can use a particular field or invoke a particular method**

Inheritance and Access Modifiers



```
public class Person {

    private long id;
    private String user;
    private String firstName;
    private String lastName;
    private Date birthDate;

    // ...
```

Encapsulation define that attributes are defined as **private**.

And private attributes cannot be inherited

So….

Inheritance and Access Modifiers

```
public class Student extends Person {
```

```
firstName has private access in sia.Person
--
(Alt-Enter shows hints)
```

```
Student student = new Student();
student.firstName = "Clark";
```

How can be accessed superclass attributes from a subclass?

Inheritance and Access Modifiers

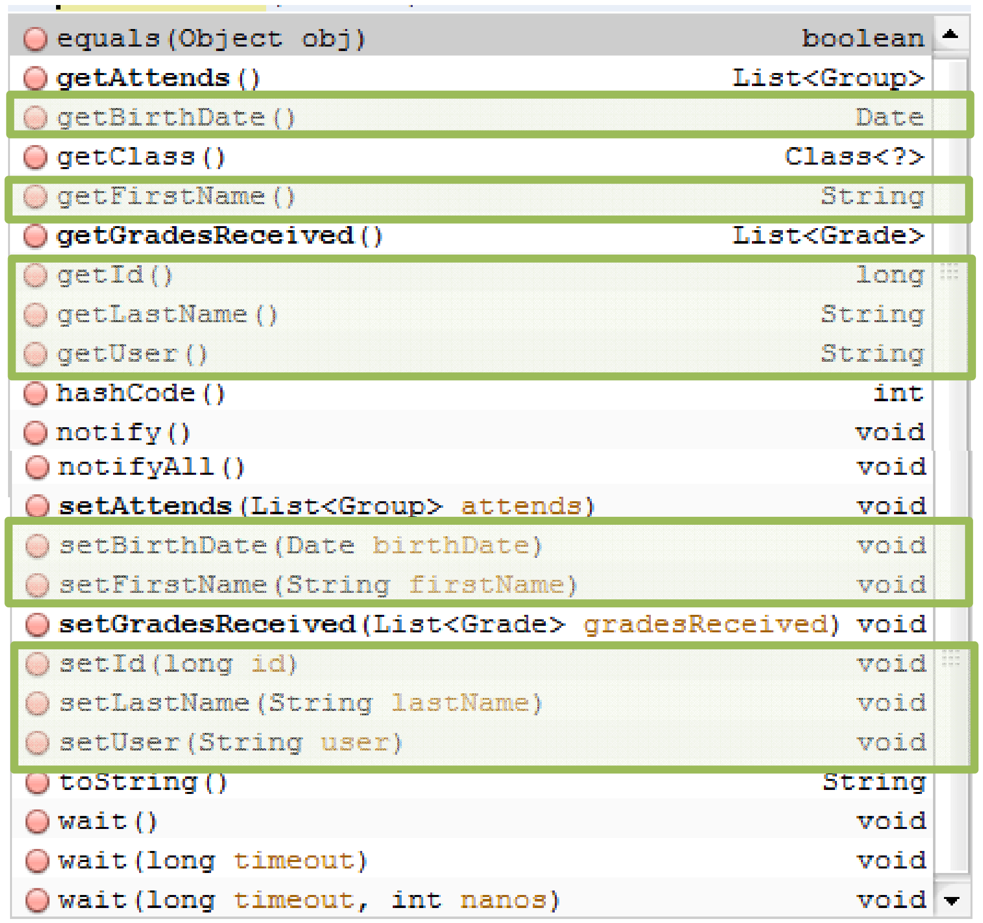
We can access to private attributes through **public superclass methods**

```
Student student = new Student();
student.setFirstName("Clark");
```

Inheritance and Access Modifiers

Public or protected Person Methods are inherited by the **Student** subclass

```
public class Student extends Person {
```



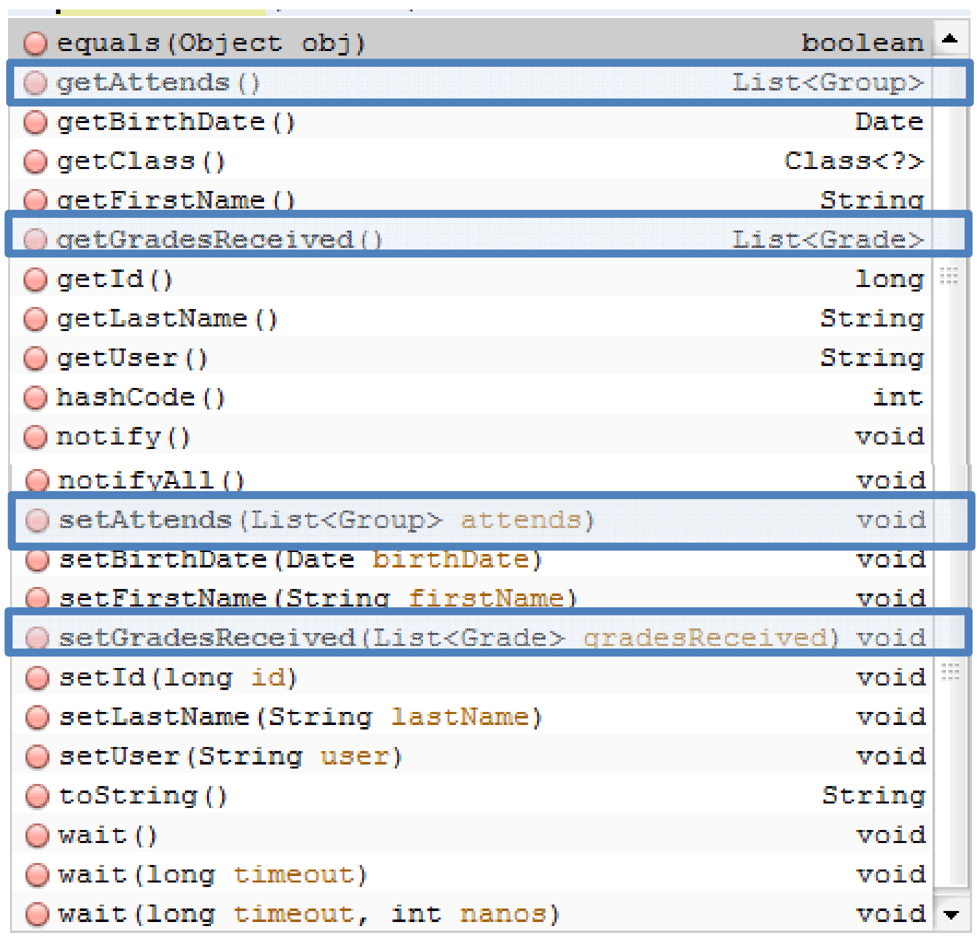
```
equals(Object obj)                                      boolean
getAttends()                                        List<Group>
getBirthDate()                                             Date
getClass()                                             Class<?>
getFirstName()                                           String
getGradesReceived()                                 List<Grade>
getId()                                                    long
getLastName()                                            String
getUser()                                                String
hashCode()                                                  int
notify()                                                   void
notifyAll()                                                void
setAttends(List<Group> attends)                            void
setBirthDate(Date birthDate)                               void
setFirstName(String firstName)                             void
setGradesReceived(List<Grade> gradesReceived)              void
setId(long id)                                             void
setLastName(String lastName)                               void
setUser(String user)                                       void
toString()                                               String
wait()                                                     void
wait(long timeout)                                         void
wait(long timeout, int nanos)                              void
```

Inherited Person methods

Inheritance and Access Modifiers

Public or protected
Person Methods are inherited by the **Student** subclass

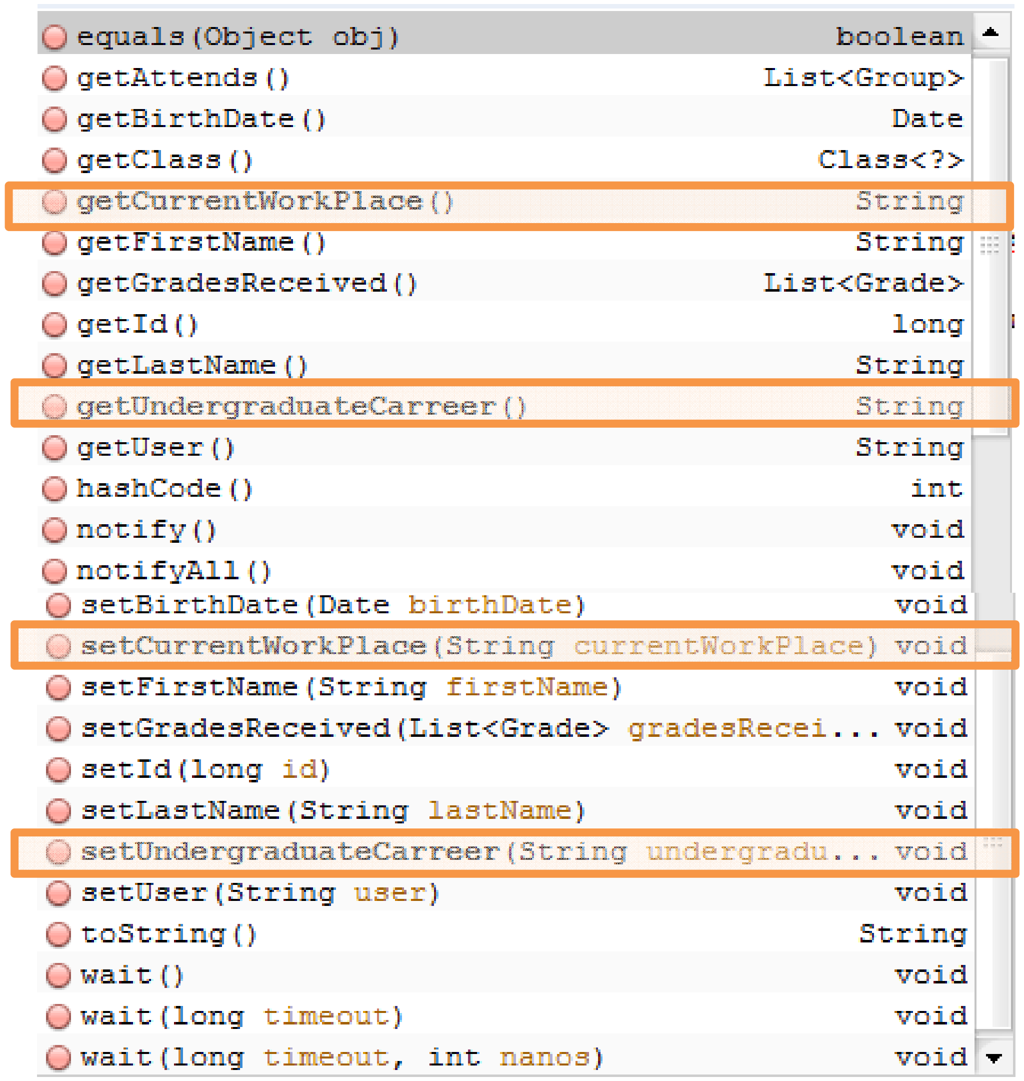
```
public class Student extends Person {
```



equals(Object obj)	boolean
getAttends()	List<Group>
getBirthDate()	Date
getClass()	Class<?>
getFirstName()	String
getGradesReceived()	List<Grade>
getId()	long
getLastName()	String
getUser()	String
hashCode()	int
notify()	void
notifyAll()	void
setAttends(List<Group> attends)	void
setBirthDate(Date birthDate)	void
setFirstName(String firstName)	void
setGradesReceived(List<Grade> gradesReceived)	void
setId(long id)	void
setLastName(String lastName)	void
setUser(String user)	void
toString()	String
wait()	void
wait(long timeout)	void
wait(long timeout, int nanos)	void

Student methods

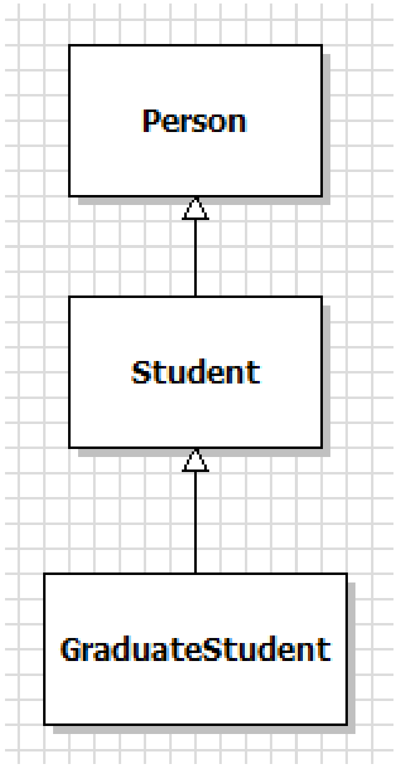
Inheritance and Access Modifiers



```
equals(Object obj)                              boolean
getAttends()                                    List<Group>
getBirthDate()                                  Date
getClass()                                      Class<?>
getCurrentWorkPlace()                           String
getFirstName()                                  String
getGradesReceived()                             List<Grade>
getId()                                         long
getLastName()                                   String
getUndergraduateCarreer()                       String
getUser()                                       String
hashCode()                                      int
notify()                                        void
notifyAll()                                     void
setBirthDate(Date birthDate)                    void
setCurrentWorkPlace(String currentWorkPlace)    void
setFirstName(String firstName)                  void
setGradesReceived(List<Grade> gradesRecei...    void
setId(long id)                                  void
setLastName(String lastName)                    void
setUndergraduateCarreer(String undergradu...    void
setUser(String user)                            void
toString()                                      String
wait()                                          void
wait(long timeout)                              void
wait(long timeout, int nanos)                   void
```

Public or protected Person and Student Methods are inherited by the **GraduateStudent** subclass

Inheritance and Access Modifiers



Public or protected
Person and Student
Methods are inherited
by the
GraduateStudent
subclass

Overriding methods

Overriding

Overriding involves “rewriting” how a method works internally, **without changing the signature** of that method.

In the real life

Animals talk in different ways

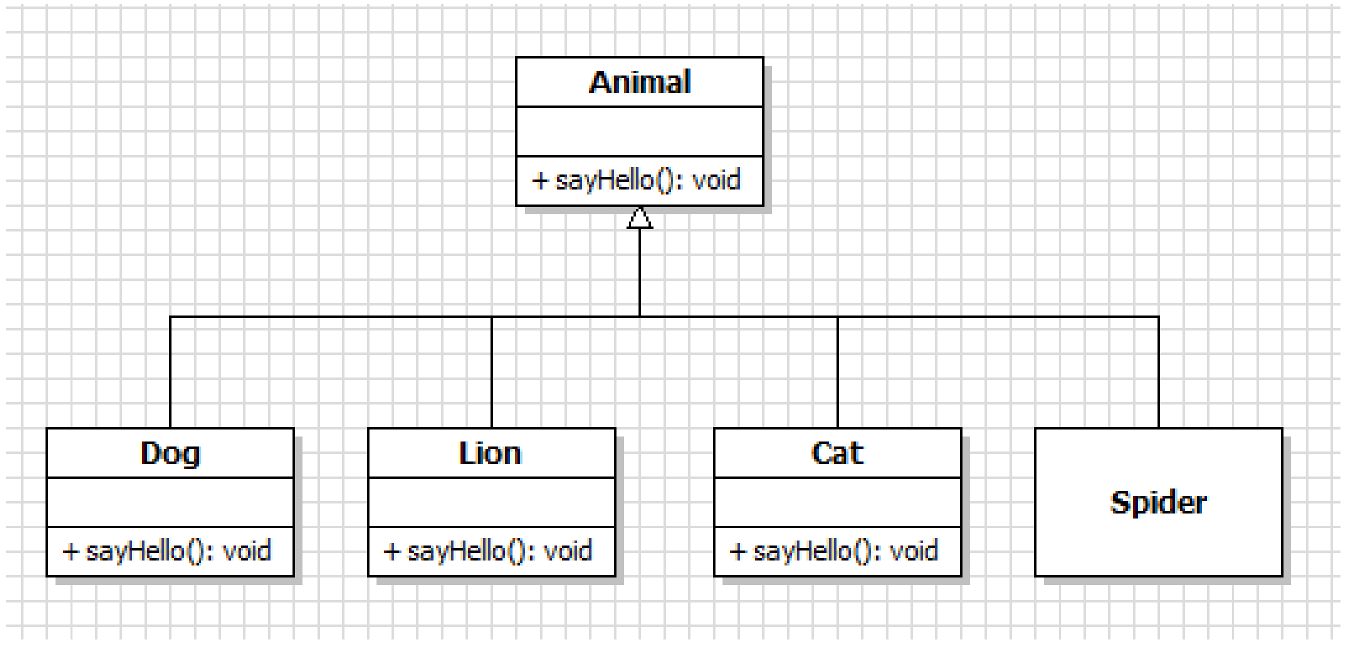
Dogs say: GUAU!!!

Cats say: MEOW!!!

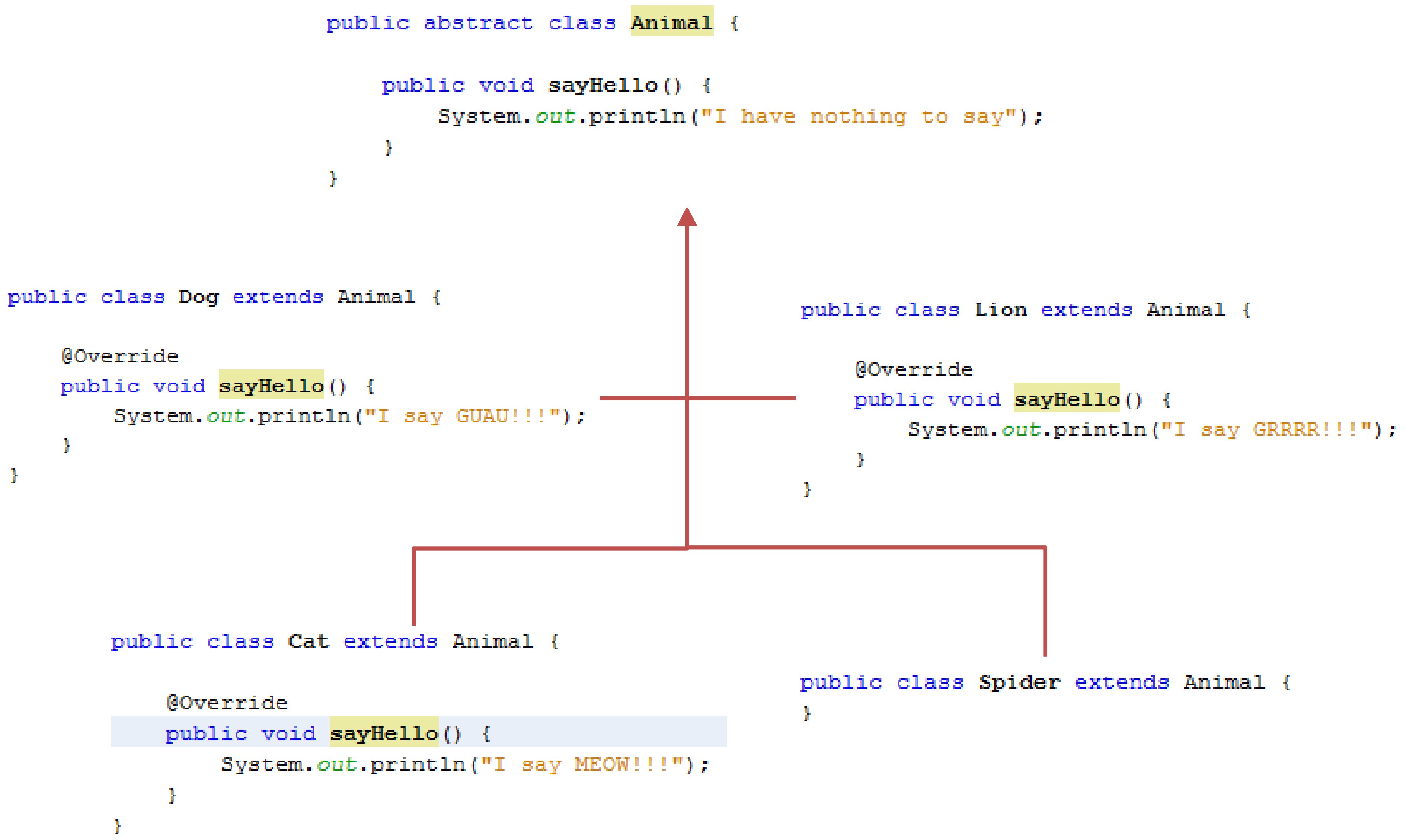
Lions say: GRRR!!!

Spiders say: ……..

In UML life


Animal
+ sayHello(): void
Dog
+ sayHello(): void
Lion
+ sayHello(): void
Cat
+ sayHello(): void
Spider

I am the superclass



```
public abstract class Animal {

    public void sayHello() {
        System.out.println("I have nothing to say");
    }
}
```

```
public class Dog extends Animal {

    @Override
    public void sayHello() {
        System.out.println("I say GUAU!!!");
    }
}
```

```
public class Lion extends Animal {

    @Override
    public void sayHello() {
        System.out.println("I say GRRRR!!!");
    }
}
```

```
public class Cat extends Animal {

    @Override
    public void sayHello() {
        System.out.println("I say MEOW!!!");
    }
}
```

```
public class Spider extends Animal {
}
```

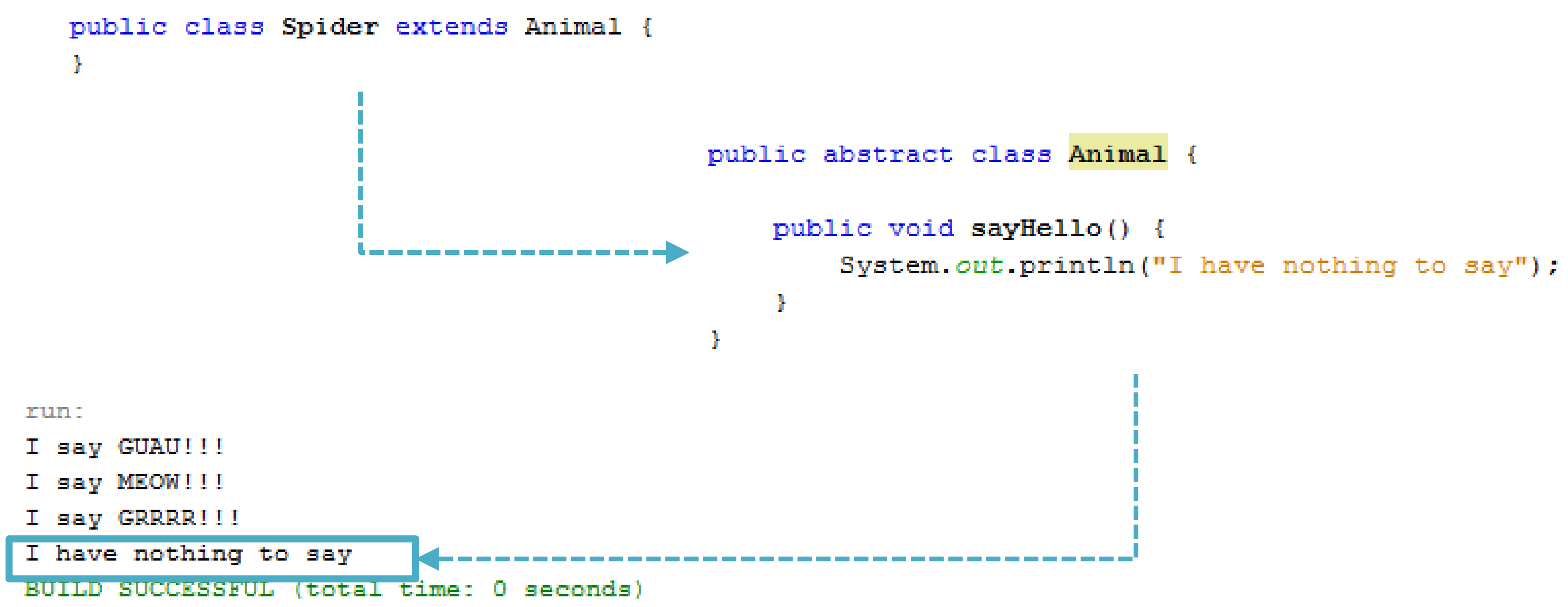
Overriding example

```java
public class ZooTest {

    public static void main(String[] args) {

        Animal dog = new Dog();
        Animal cat = new Cat();
        Animal lion = new Lion();
        Animal spider = new Spider();

        dog.sayHello();
        cat.sayHello();
        lion.sayHello();
        spider.sayHello();

    }
}
```

Respective override `sayHello` method is called

```
run:
I say GUAU!!!
I say MEOW!!!
I say GRRRR!!!
I have nothing to say
BUILD SUCCESSFUL (total time: 0 seconds)
```

Overriding example



```
public class Spider extends Animal {
}
```

```
public abstract class Animal {

    public void sayHello() {
        System.out.println("I have nothing to say");
    }
}
```

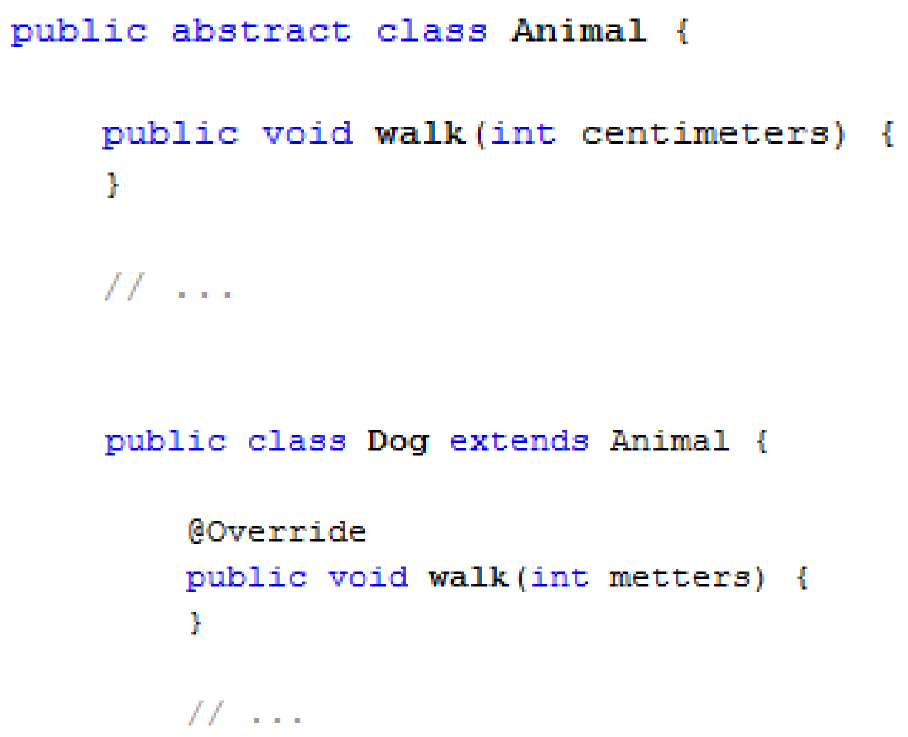
```
run:
I say GUAU!!!
I say MEOW!!!
I say GRRRR!!!
I have nothing to say
BUILD SUCCESSFUL (total time: 0 seconds)
```

If no method is found, the JVM search for it in the superclass

Overriding example (another method)

This is a **valid override**, because both methods have the same signature

walk (int)



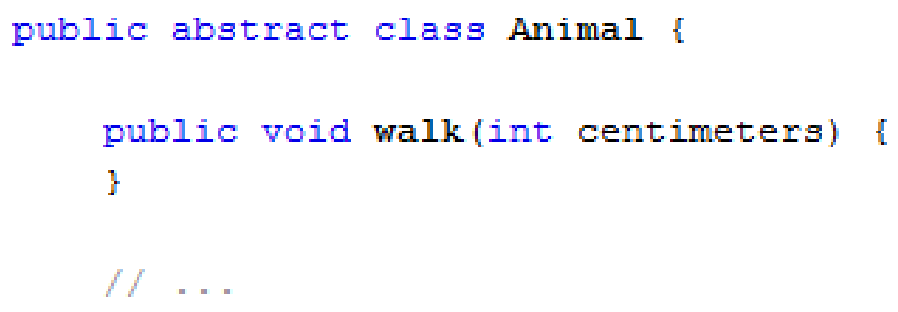
```
public abstract class Animal {

    public void walk(int centimeters) {
    }

    // ...


    public class Dog extends Animal {

        @Override
        public void walk(int metters) {
        }

        // ...
```

Overriding example (another method)

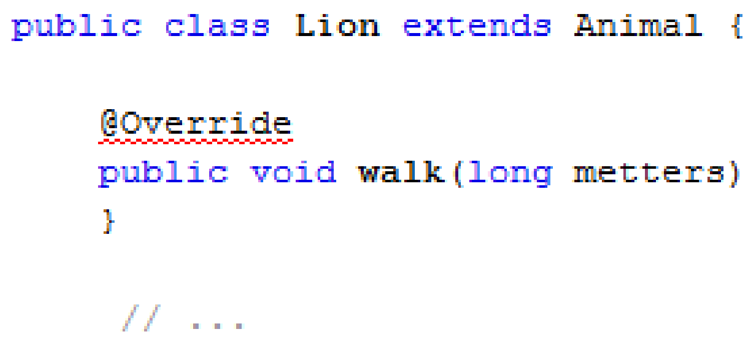
This is a **invalid override**, because both methods have different signature

walk (int)

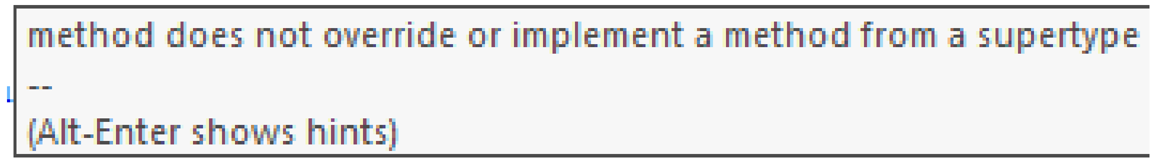
walk (long)



```
public abstract class Animal {

    public void walk(int centimeters) {
    }

    // ...
```



```
public class Lion extends Animal {

    @Override
    public void walk(long metters) {
    }

     // ...
```

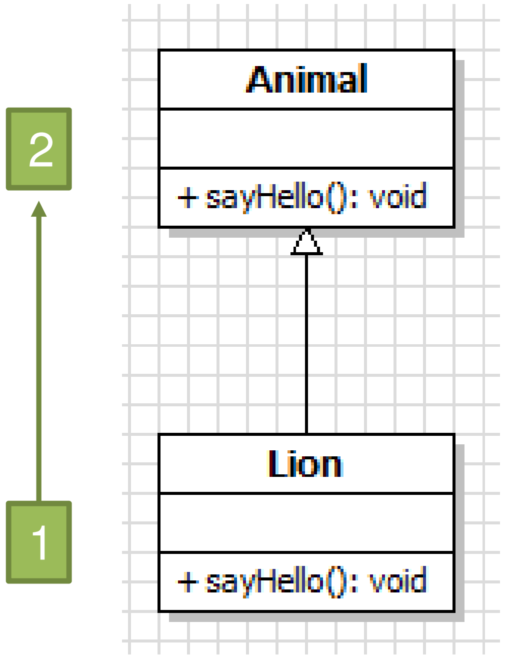


method does not override or implement a method from a supertype
--
(Alt-Enter shows hints)

Execution order

When we override methods, first is called the more specific method

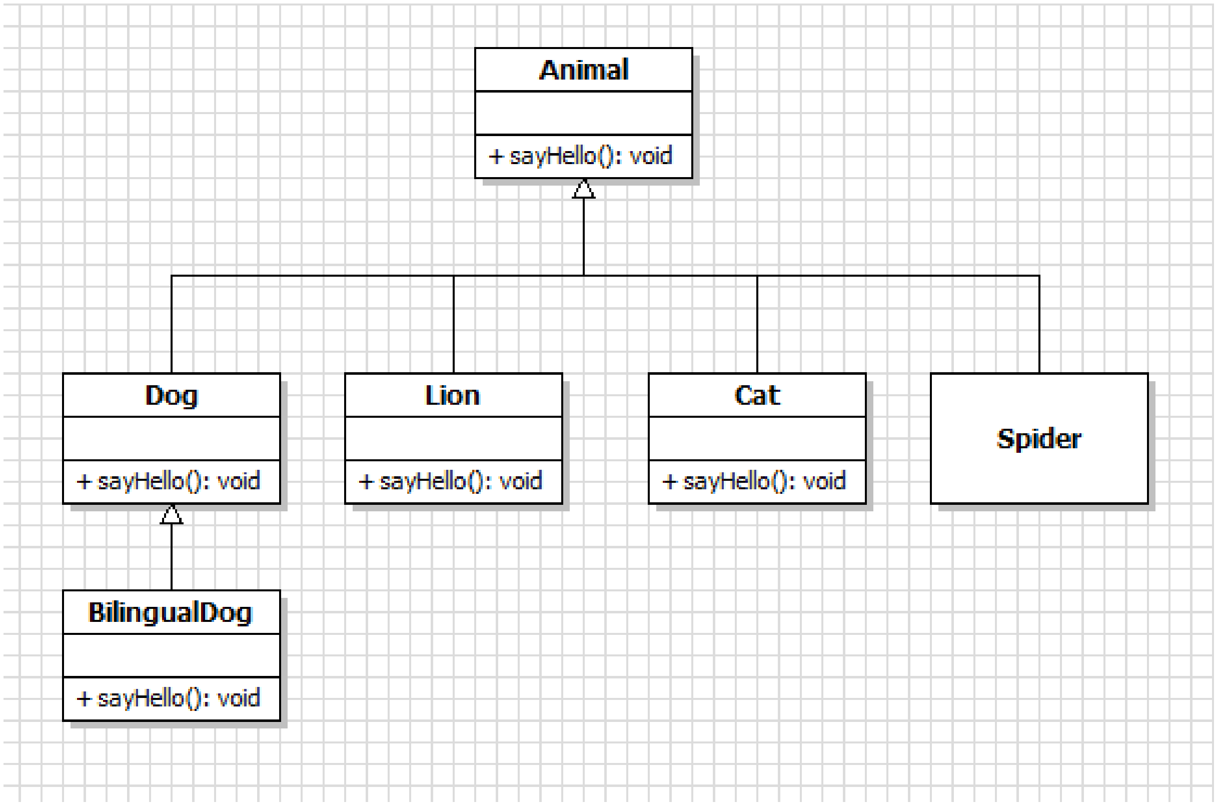
More general



More specific

Reusing superclass behaviors

Reusing superclass methods



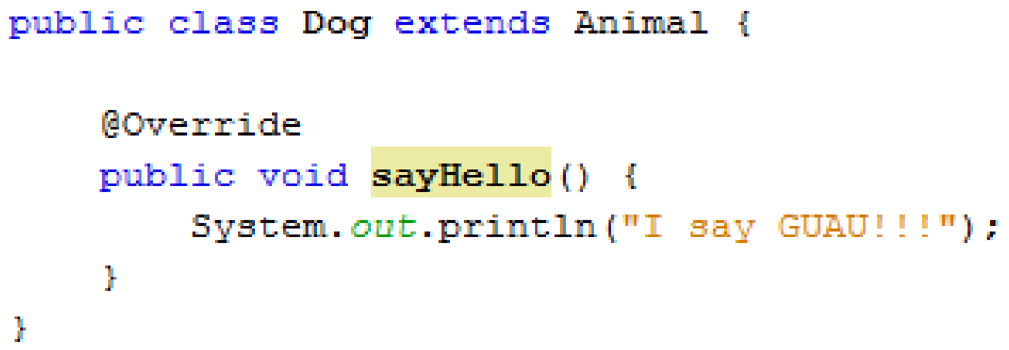
Bilingual Dog can say GUAU!!!

Bilingual Dog can say REGUAUSS!!!

Normal Dog

Normal Dog only say GUAU!!!

Bilingual Dog speaks normal **Dog Language** and **Ancient Dog Language**



```
public class Dog extends Animal {

    @Override
    public void sayHello() {
        System.out.println("I say GUAU!!!");
    }
}
```

We can reuse the superclass methods with `super` keyword

```
public class BilingualDog extends Dog {

    @Override
    public void sayHello() {
        super.sayHello();
        System.out.println("I say REGUAUSS!!!");
    }
}
```

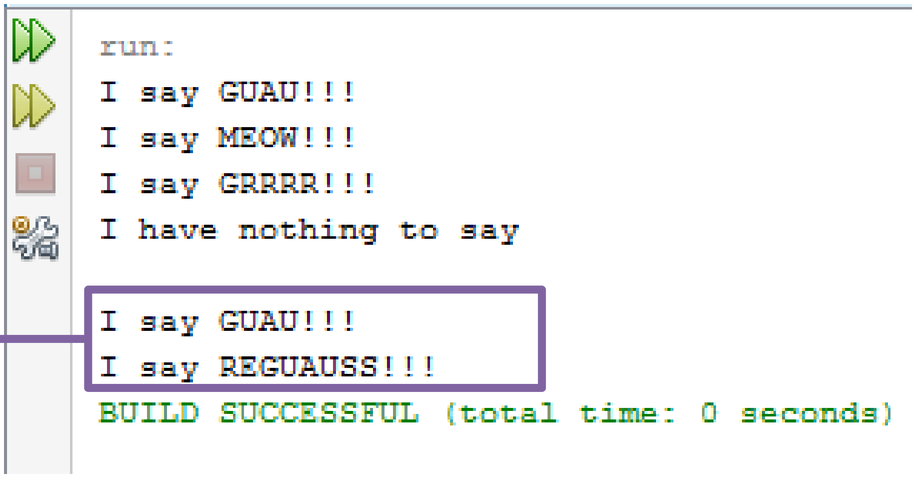
Bilingual dog can speak as normal dog using the same methods

I am reusing the superclass dog sayHello() method

Reusing superclass methods

```java
public class ZooTest {

    public static void main(String[] args) {

        Animal dog = new Dog();
        Animal cat = new Cat();
        Animal lion = new Lion();
        Animal spider = new Spider();
        Animal biligualDog = new BilingualDog();

        dog.sayHello();
        cat.sayHello();
        lion.sayHello();
        spider.sayHello();

        System.out.println();

        biligualDog.sayHello();

    }
}
```

```java
public class BilingualDog extends Dog {

    @Override
    public void sayHello() {

        super.sayHello();

        System.out.println("I say REGUAUSS!!!");
    }
}
```



```
run:
I say GUAU!!!
I say MEOW!!!
I say GRRRR!!!
I have nothing to say

I say GUAU!!!
I say REGUAUSS!!!
BUILD SUCCESSFUL (total time: 0 seconds)
```

Time to play in the pet store

1. Using UML design a **class hierarchy** (at least 3 levels of inheritance) for a **pet store with** at least 6 different kind of pets

2. Create the **Java classes definitions** (encapsulated) for the pets available on the pet store, each pet must have at least 3 attributes (not inherited).

3. Create a **test class** for your pet store, this class must show a menu with the available pets (previously created).

4. User can select a pet and the **system must show all information available for this pet (including ancestor information).**

5. Program finish when user select the option finish in the main menu

6. **You have to use the keyword super and override annotation.**

References

[Barker] J. Barker, *Beginning Java Objects: From Concepts To Code*, Second Edition, Apress, 2005.

[Oracle] *Understanding Instance and Class Members,* Available: http://download.oracle.com/javase/tutorial/java/javaOO/classvars.html

[Oracle] Java API documentation, *Class Object,* Available: http://download.oracle.com/javase/6/docs/api/java/lang/Object.html