Java (Core Java)

Abstraction

Today’s Topics

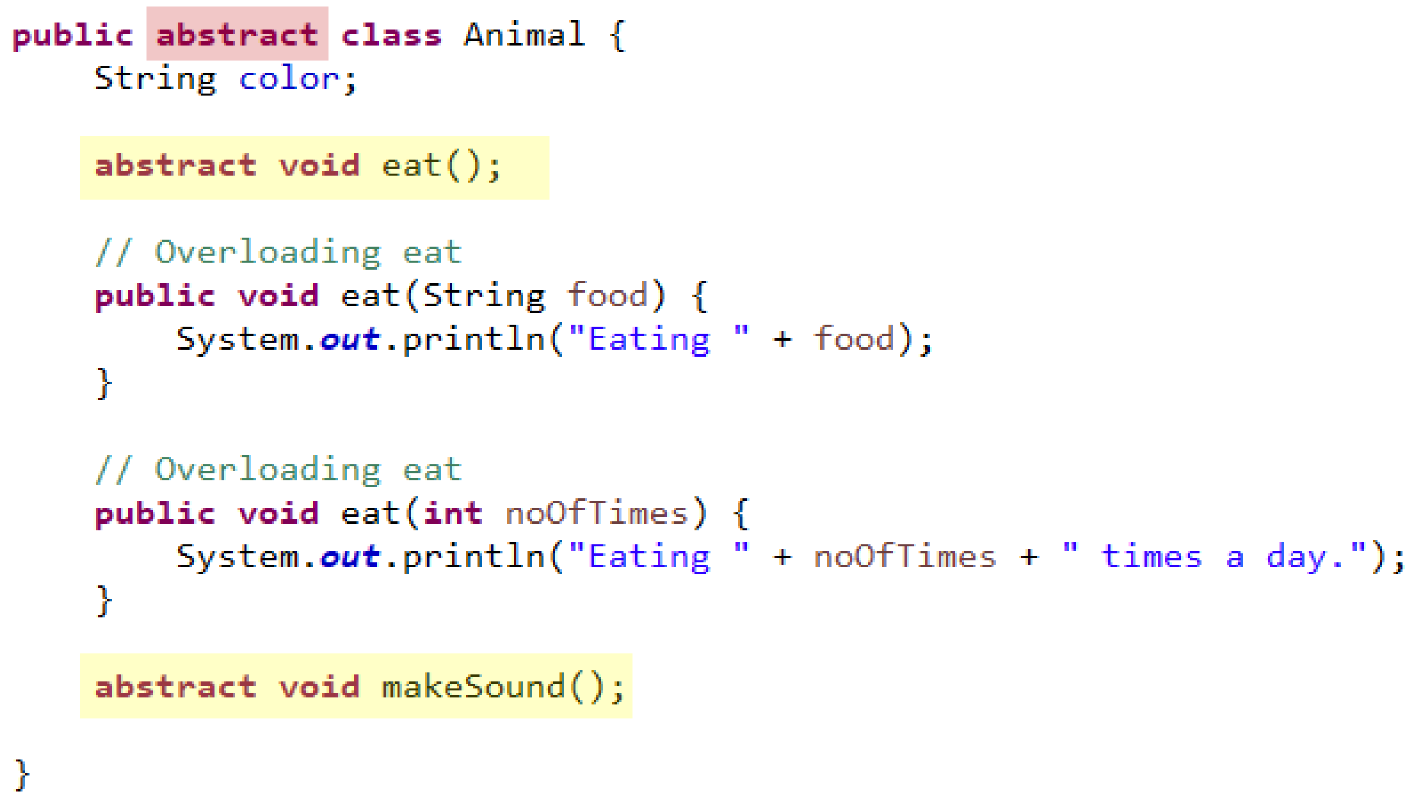
- Abstraction
  - Abstract Class
  - Inheritance and Overriding related to abstract classes
  - Interface and Implementation

Abstraction

- Hide the implementation details
- Show only functionality to the user
- Abstraction can be partial (just one, or multiple method(s) in a class) or complete (all methods are abstracted)
- A class which has at least one abstract method needs to be defined as abstract
- Objects cannot be instantiated for abstract classes. To use it, some other class must extend the abstract class, or implement the interface.

Let us understand all this in details.

Example of an abstract class



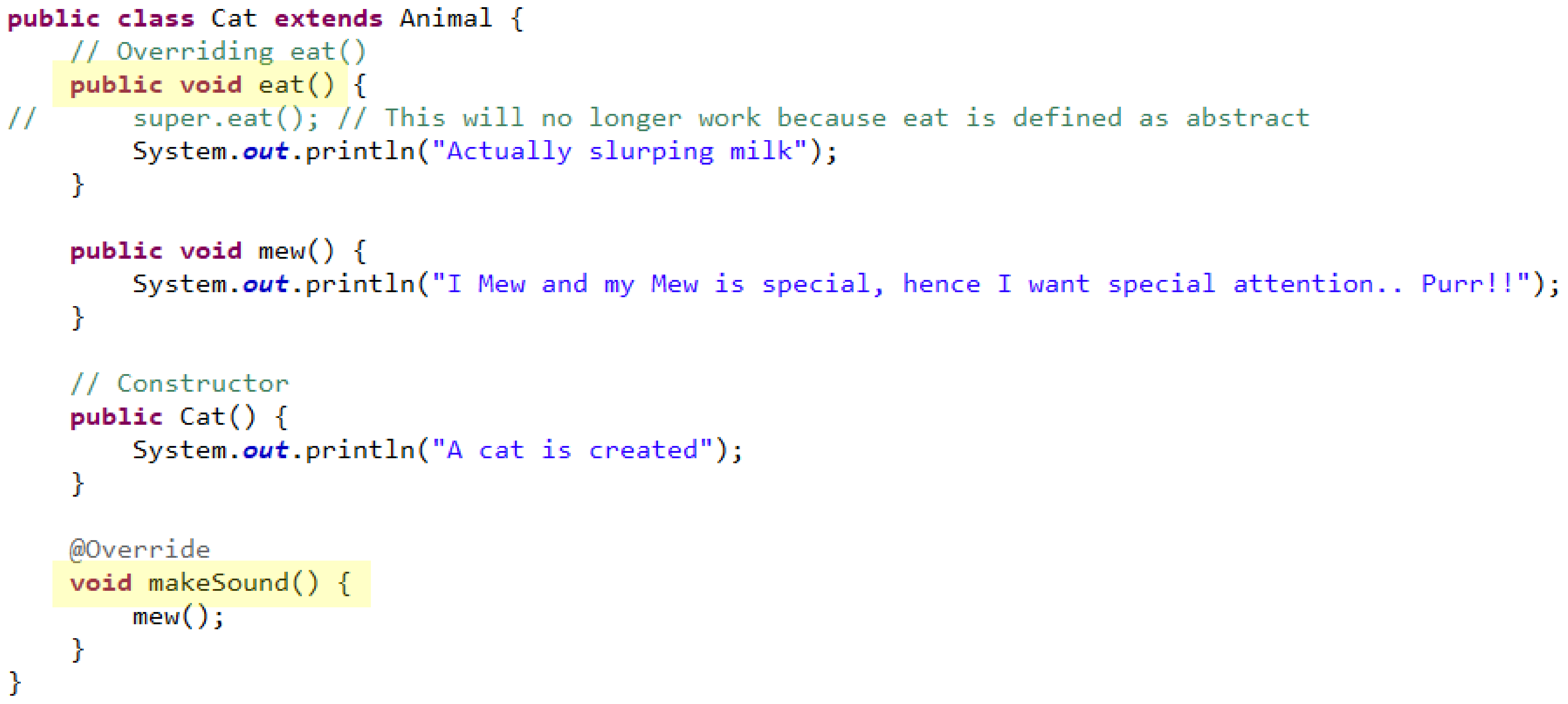
```
public abstract class Animal {
    String color;

    abstract void eat();

    // Overloading eat
    public void eat(String food) {
        System.out.println("Eating " + food);
    }

    // Overloading eat
    public void eat(int noOfTimes) {
        System.out.println("Eating " + noOfTimes + " times a day.");
    }

    abstract void makeSound();

}
```

- eat() and makeSound() are declared but no implementation. Hence called out as abstract
- Since there are abstract methods in the class, hence class also becomes abstract
- The moment class is abstract, you cannot create instance of Animal any more, and hence it mandates you to extend the class

Utility of abstraction

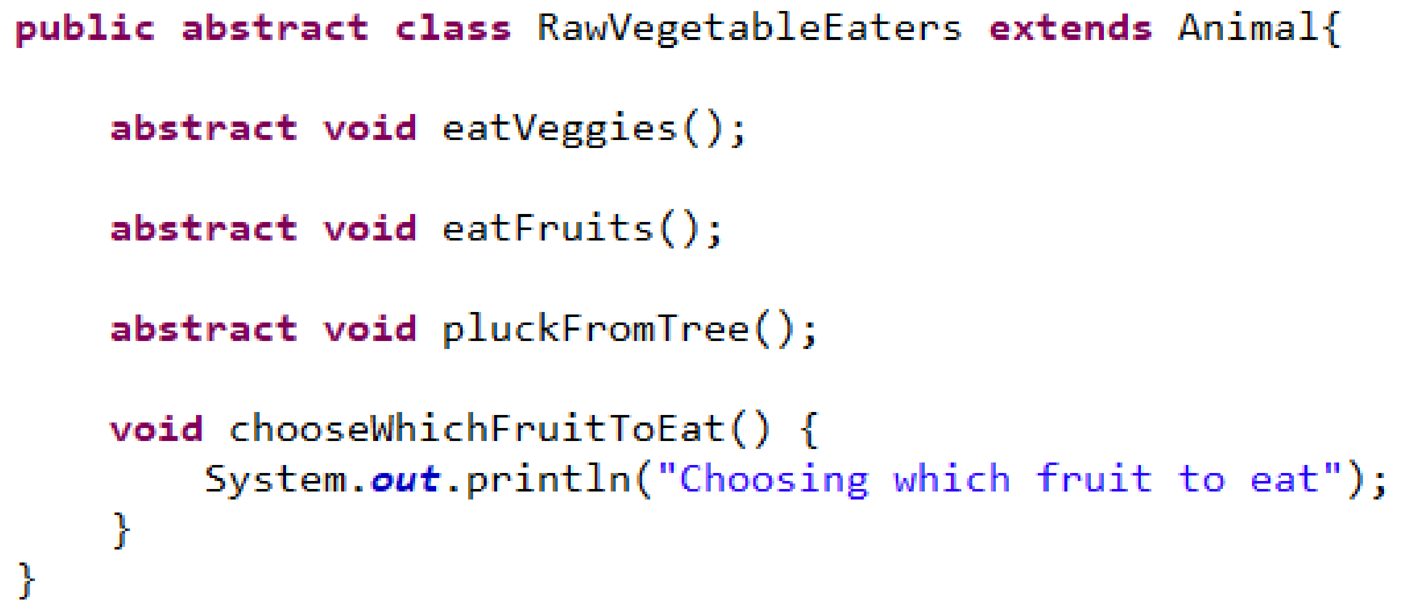
- Abstraction helps you define a template. For instance, when you define an Animal, you focus only on its traits and not their details.
  - An Animal moves, eats, drinks, breathes, sleeps… These are traits of an animal.
  - But why bother putting implementation in Animal class? Since all animals will never eat, move, breathe, or even sleep the same way

Implementation of abstracted method is done through overriding.



```java
public class Cat extends Animal {
    // Overriding eat()
    public void eat() {
//      super.eat(); // This will no longer work because eat is defined as abstract
        System.out.println("Actually slurping milk");
    }

    public void mew() {
        System.out.println("I Mew and my Mew is special, hence I want special attention.. Purr!!");
    }

    // Constructor
    public Cat() {
        System.out.println("A cat is created");
    }

    @Override
    void makeSound() {
        mew();
    }
}
```

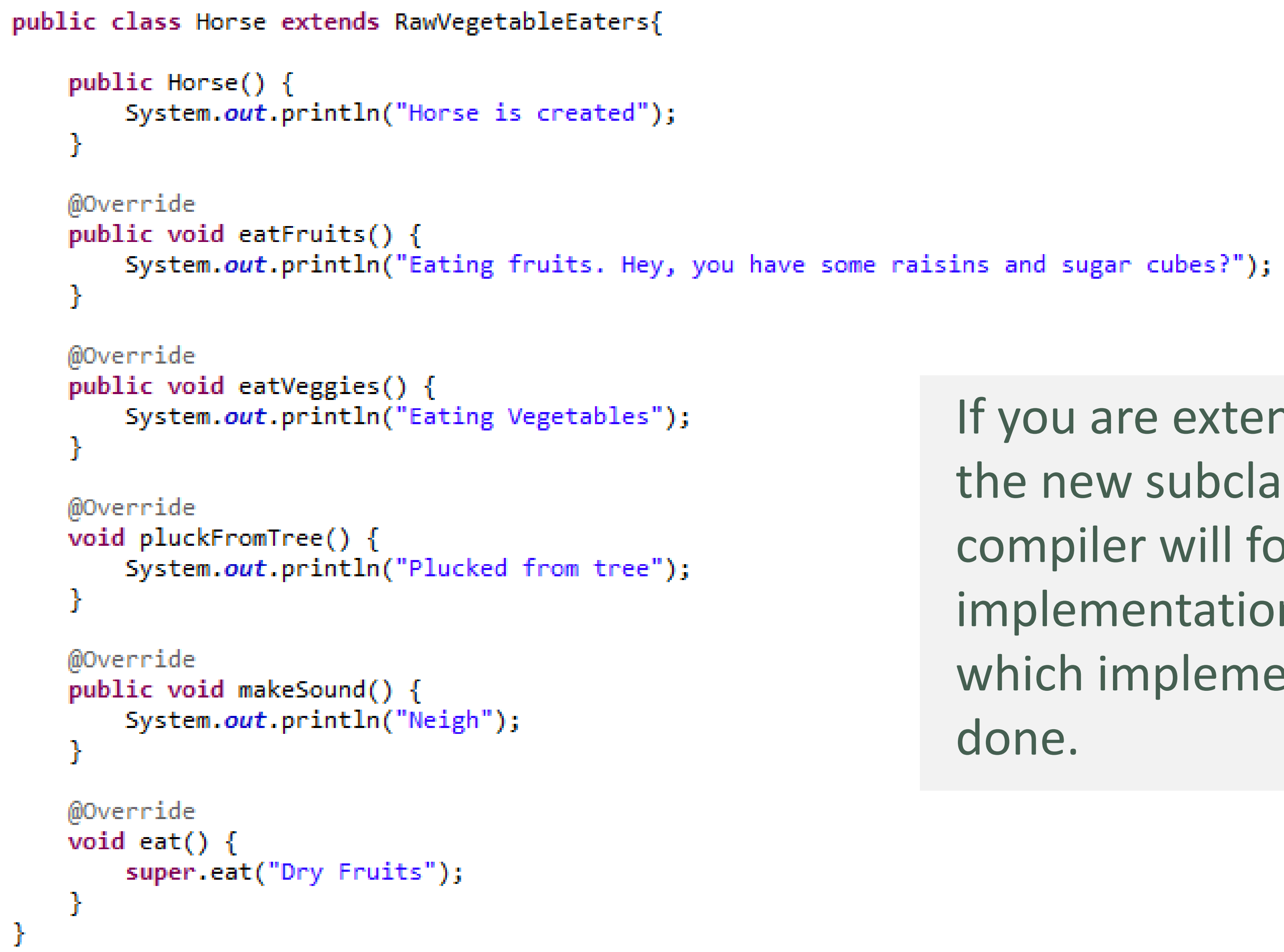
The class that extends abstract class can also be abstract



```java
public abstract class RawVegetableEaters extends Animal{

    abstract void eatVeggies();

    abstract void eatFruits();

    abstract void pluckFromTree();

    void chooseWhichFruitToEat() {
        System.out.println("Choosing which fruit to eat");
    }
}
```

- Abstraction is part of design. Hence, Java allows you multiple levels of abstraction across inheritance hierarchies
- By the time the first non abstract class is encountered in the inheritance chain, implementation of all abstract methods should be available, so that objects can be created.

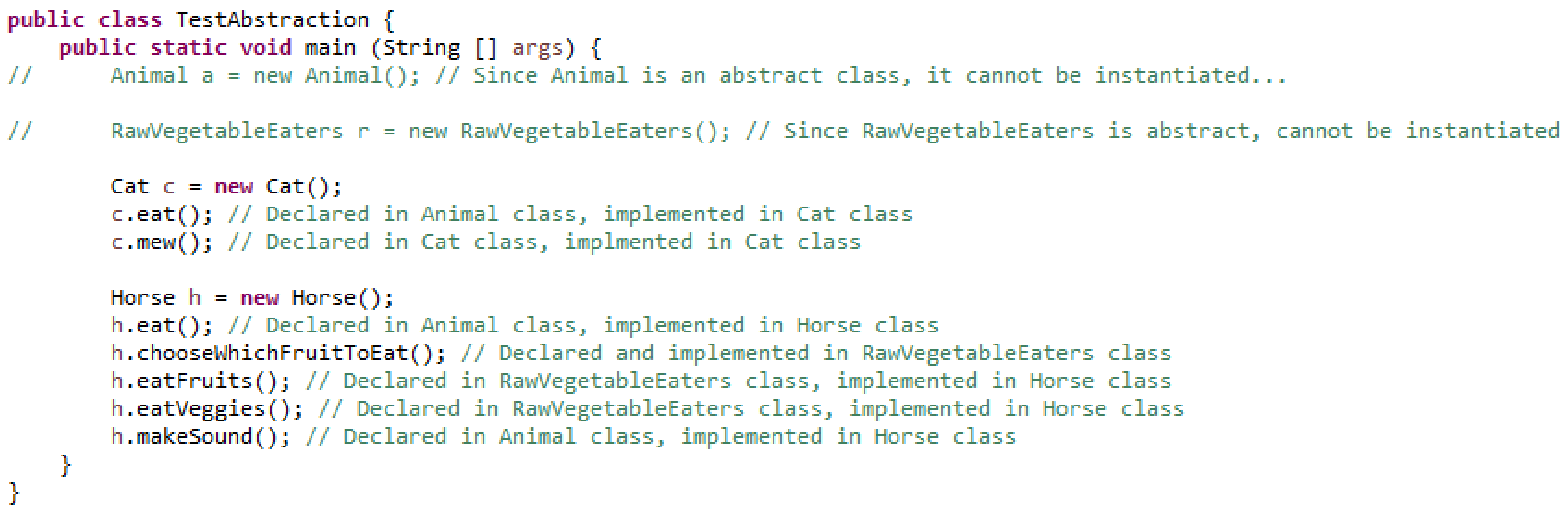
Extending an abstract class



```java
public class Horse extends RawVegetableEaters{

    public Horse() {
        System.out.println("Horse is created");
    }

    @Override
    public void eatFruits() {
        System.out.println("Eating fruits. Hey, you have some raisins and sugar cubes?");
    }

    @Override
    public void eatVeggies() {
        System.out.println("Eating Vegetables");
    }

    @Override
    void pluckFromTree() {
        System.out.println("Plucked from tree");
    }

    @Override
    public void makeSound() {
        System.out.println("Neigh");
    }

    @Override
    void eat() {
        super.eat("Dry Fruits");
    }
}
```

If you are extending the abstract class, and the new subclass is non abstract, then compiler will force you to put implementation for all those methods for which implementation has not yet been done.

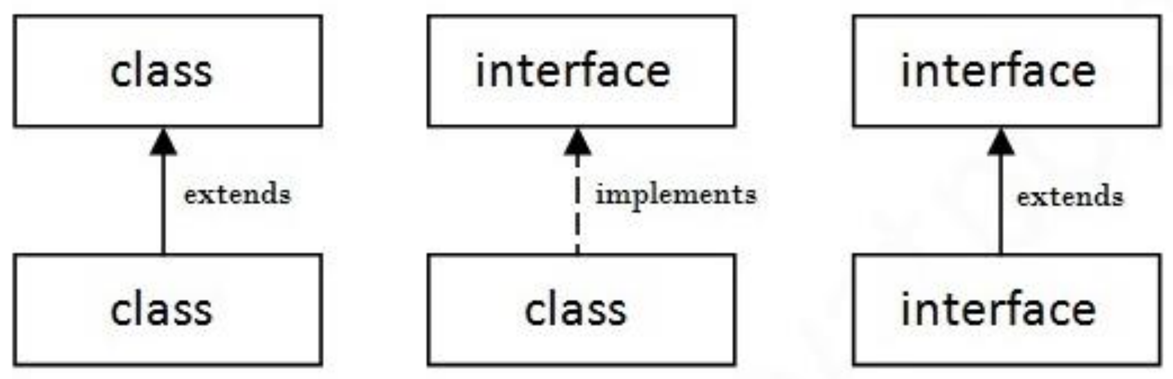
Instantiating objects – not allowed for abstract classes



```
public class TestAbstraction {
    public static void main (String [] args) {
//      Animal a = new Animal(); // Since Animal is an abstract class, it cannot be instantiated...

//      RawVegetableEaters r = new RawVegetableEaters(); // Since RawVegetableEaters is abstract, cannot be instantiated

        Cat c = new Cat();
        c.eat(); // Declared in Animal class, implemented in Cat class
        c.mew(); // Declared in Cat class, implmented in Cat class

        Horse h = new Horse();
        h.eat(); // Declared in Animal class, implemented in Horse class
        h.chooseWhichFruitToEat(); // Declared and implemented in RawVegetableEaters class
        h.eatFruits(); // Declared in RawVegetableEaters class, implemented in Horse class
        h.eatVeggies(); // Declared in RawVegetableEaters class, implemented in Horse class
        h.makeSound(); // Declared in Animal class, implemented in Horse class
    }
}
```

100% abstraction - Interface

- Every method in an interface is abstract
- No logic implementation in any method at all.
- Interface is also an IS-A relationship, similar to Inheritance, but while inheritance focused on attributes and methods, interface focusses more on methods



Interface example

```java
public interface Herbivorous {
    void eatFruits();
    void eatVeggies();
}
```

Interface only has the method declaration and no implementation

```java
public abstract class RawVegetableEaters extends Animal implements Herbivorous{
    abstract void pluckFromTree();

    void chooseWhichFruitToEat() {
        System.out.println("Choosing which fruit to eat");
    }
}
```

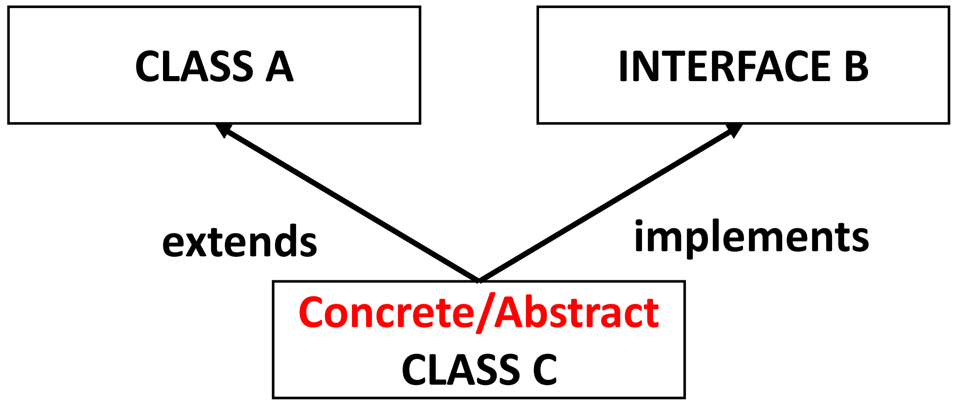
If an abstract class implements the interface, then it does not have to provide an implementation of the methods declared in the interface.

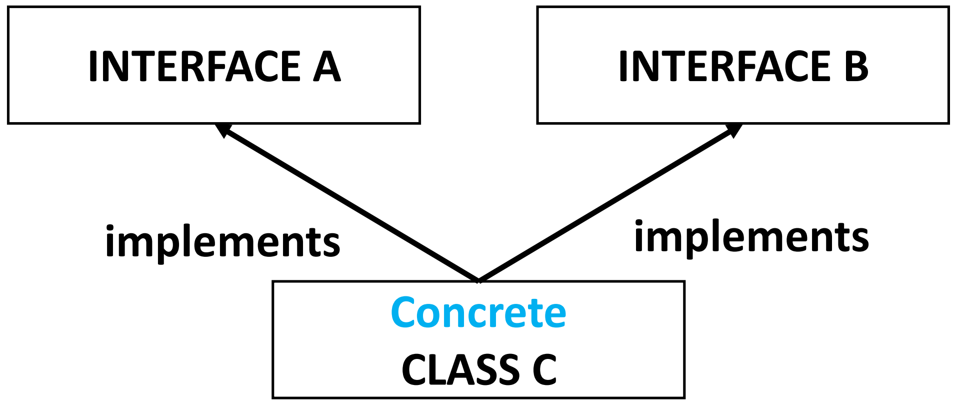
Interface example

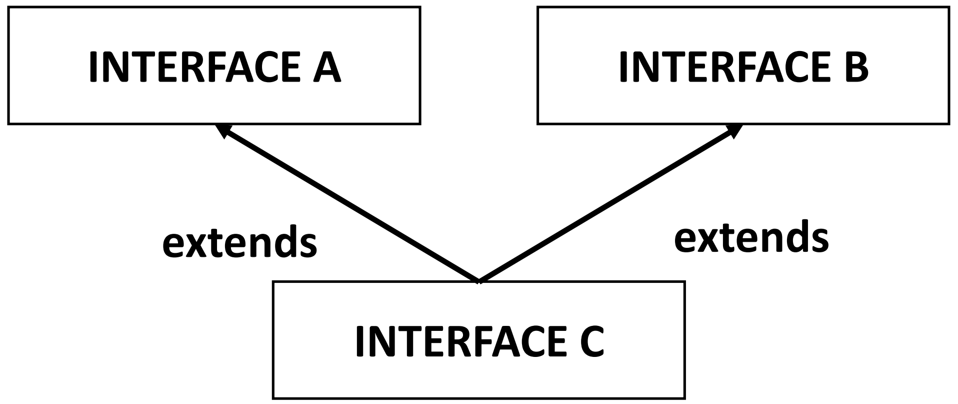
```java
public class Horse extends RawVegetableEaters{

    public Horse() {
        System.out.println("Horse is created");
    }

    @Override
    public void eatFruits() {
        System.out.println("Eating fruits. Hey, you have some raisins and sugar cubes?");
    }

    @Override
    public void eatVeggies() {
        System.out.println("Eating Vegetables");

    }

    @Override
    void pluckFromTree() {
        System.out.println("Plucked from tree");
    }
}
```

The first concrete (non abstract) class to implement the interface needs to override each of the methods of the interface and provide implementation.

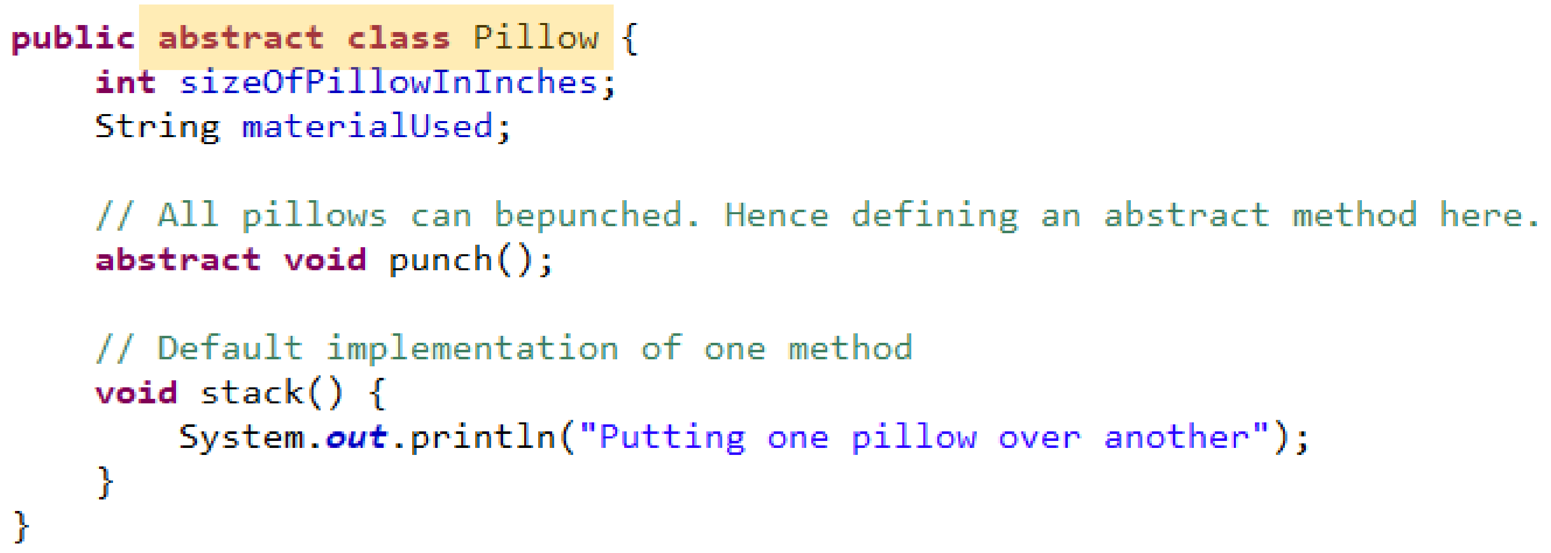
Multiple Inheritance using interfaces



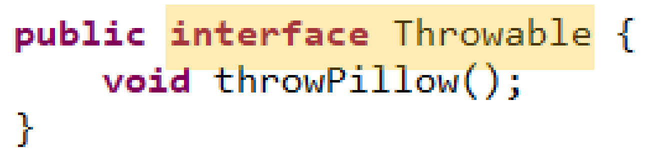




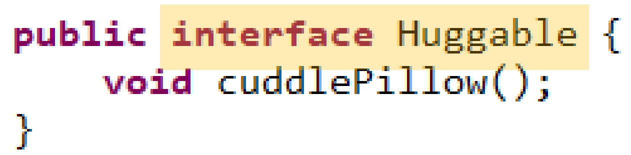
An example



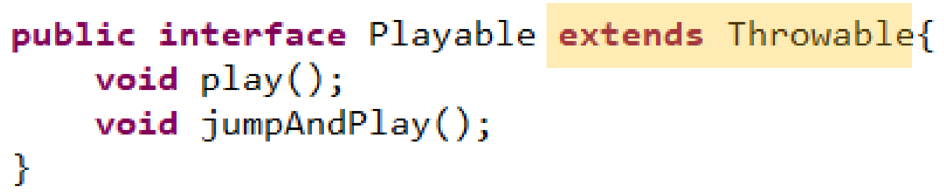
```java
public abstract class Pillow {
    int sizeOfPillowInInches;
    String materialUsed;

    // All pillows can bepunched. Hence defining an abstract method here.
    abstract void punch();

    // Default implementation of one method
    void stack() {
        System.out.println("Putting one pillow over another");
    }
}
```



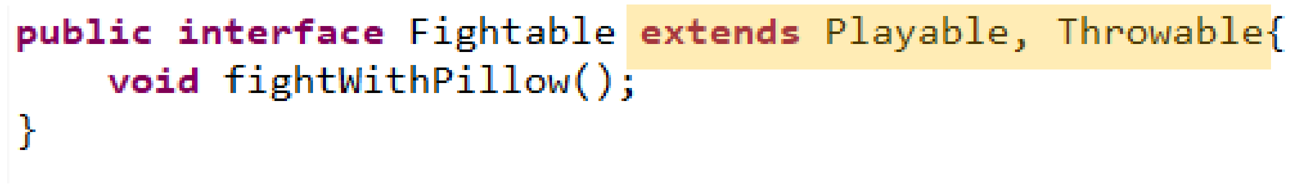
```java
public interface Throwable {
    void throwPillow();
}
```



```java
public interface Huggable {
    void cuddlePillow();
}
```

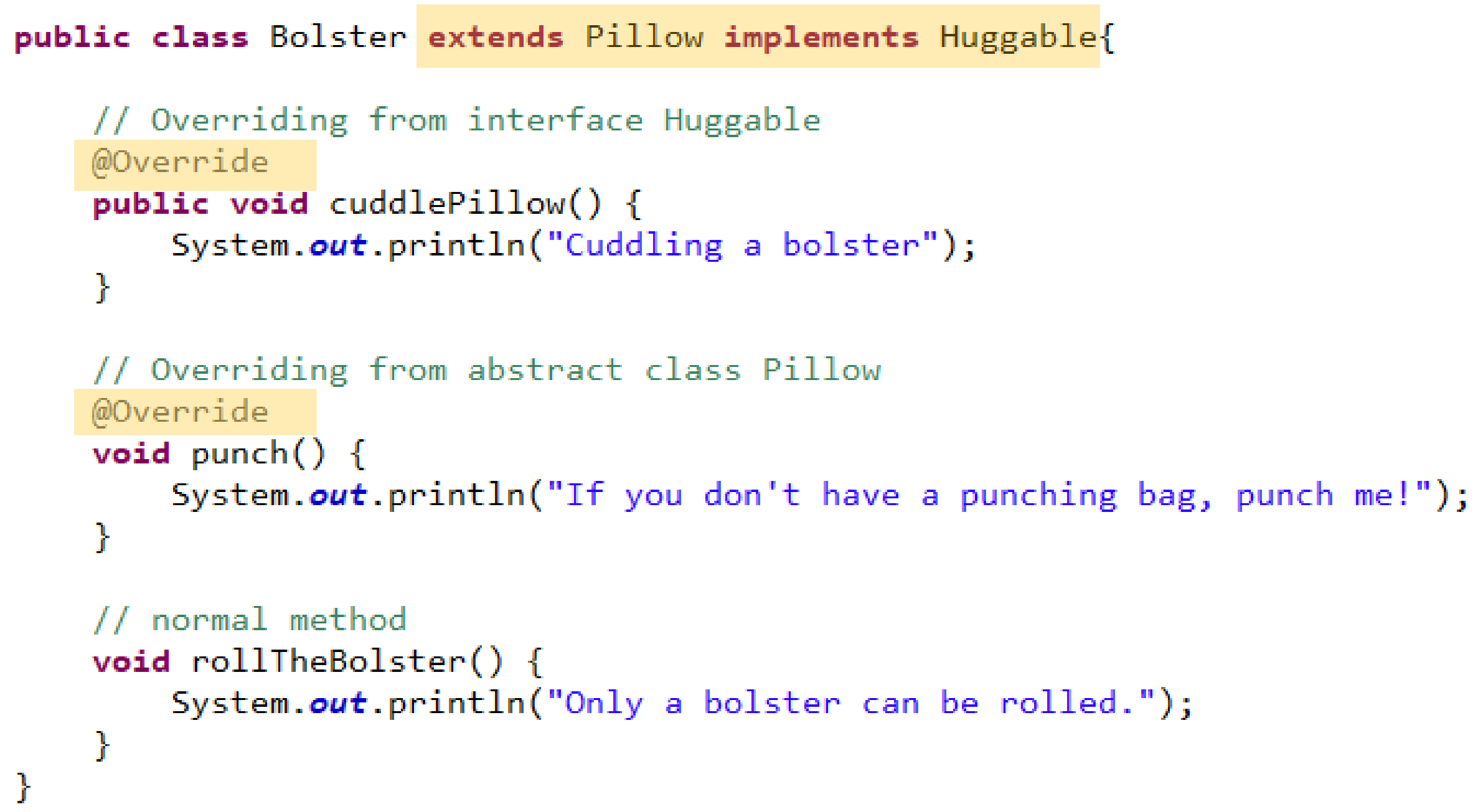


```java
public interface Playable extends Throwable{
    void play();
    void jumpAndPlay();
}
```



```java
public interface Fightable extends Playable, Throwable{
    void fightWithPillow();
}
```

An example



```
public class Bolster extends Pillow implements Huggable{

    // Overriding from interface Huggable
    @Override
    public void cuddlePillow() {
        System.out.println("Cuddling a bolster");
    }

    // Overriding from abstract class Pillow
    @Override
    void punch() {
        System.out.println("If you don't have a punching bag, punch me!");
    }

    // normal method
    void rollTheBolster() {
        System.out.println("Only a bolster can be rolled.");
    }
}
```

An example

```
public class Cushion extends Pillow implements Playable{

    @Override
    public void throwPillow() {
        System.out.println("Easy to throw me");
    }

    @Override
    public void play() {
        System.out.println("Which game you want to play? Pass pass or catch catch?");
    }

    @Override
    public void jumpAndPlay() {
        System.out.println("Seriously? Be careful not to jump on me.");
    }

    @Override
    void punch() {
        System.out.println("Spare me please. I am so small.");
    }

    void putOnSofa() {
        System.out.println("Yes, thats me. Can't put anyone else here.");
    }
}
```