

ICS 141

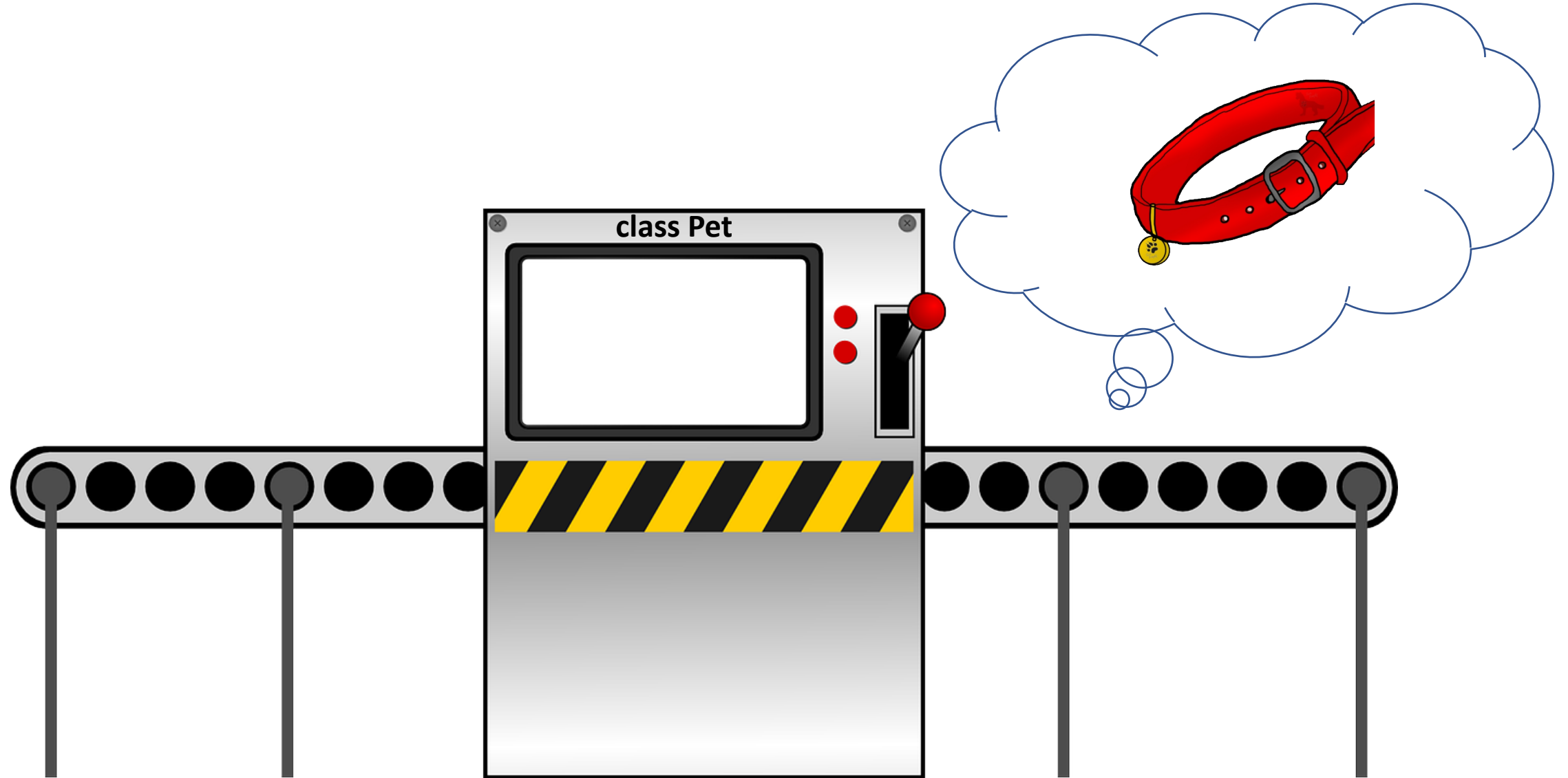
Programming with Objects

Jessica Maistrovich
Metropolitan State University

Inheritance

Extending a class

A class is a machine that can make things



```

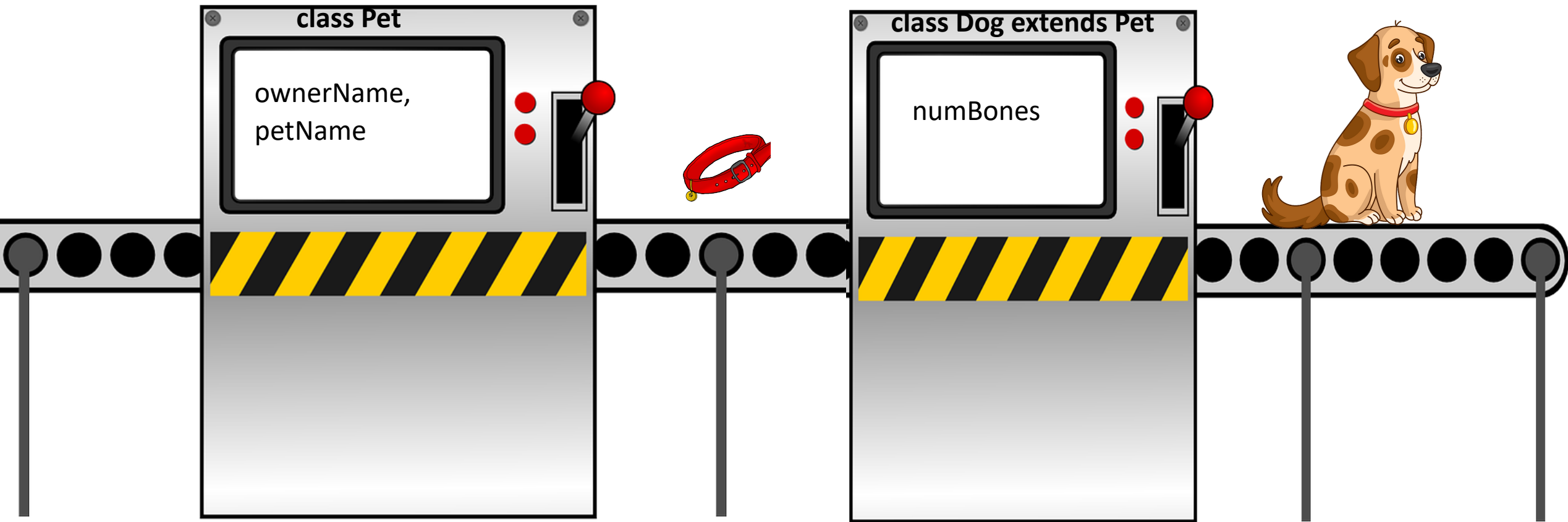
1 public class Pet {
2     private String ownerName;
3     private String petName;
4     private int petId;
5     private static int numberOfPets = 0;
6
7     public Pet(String oName, String pName) {
8         ownerName = oName;
9         petName = pName;
10        petId = ++numberOfPets;
11    }
12
13    public void setOwnerName(String oName) {
14        ownerName = oName;
15    }
16
17    public String getOwnerName() {
18        return ownerName;
19    }
20
21    public void setPetName(String pName) {
22        petName = pName;
23    }
24
25    public String getPetName() {
26        return petName;
27    }
28
29    public int getPetId() {
30        return petId;
31    }
32
33    public void speak() {
34        System.out.println("Pet noises");
35    }
36
37    public String toString() {
38        String output = new String(petName + " owned by " + ownerName + " has pet id number " + petId);
39        return output;

```

Pet

- ownerName : String
 - petName : String
 - petId : int
 - numberOfPets : int

+ Pet (String, String)
 + setOwnerName(String) : void
 + getOwnerName() : String
 + setPetName(String) : void
 + getPetName() : String
 + getPetId() : int
 + speak() : void
 + toString() : String



Dog

- numBones : int

+ Dog (String, String, int)

+ setNumBones(int) : void

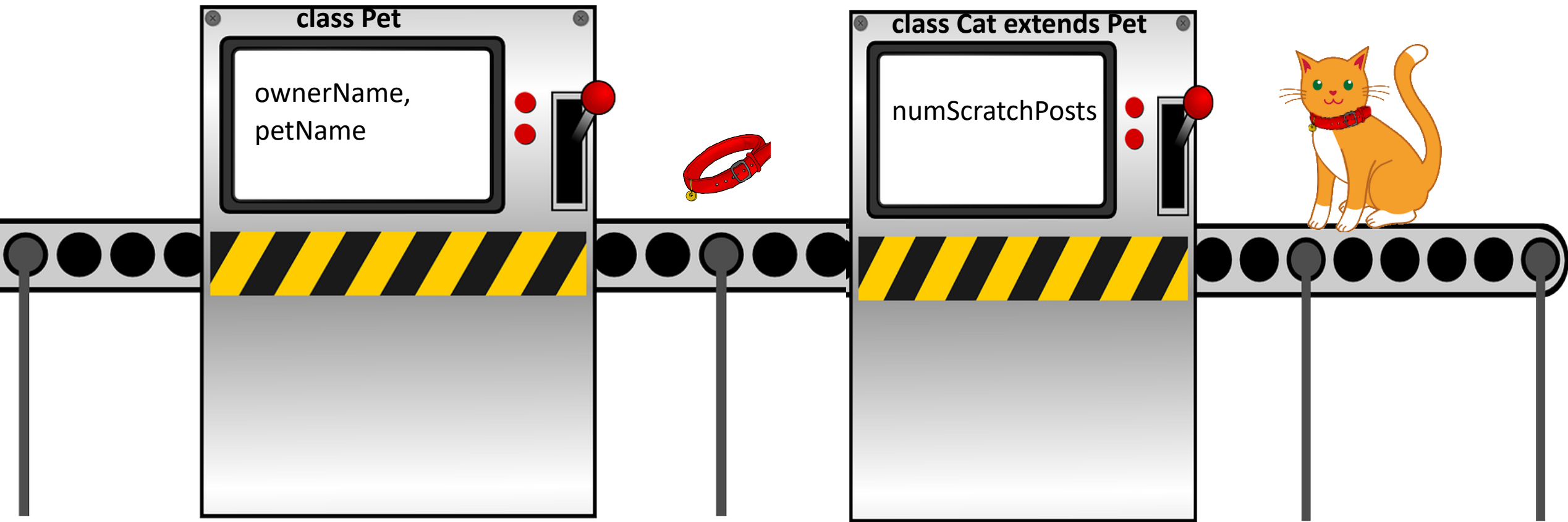
+ getNumBones() : int

+ speak() : void

+ toString() : String

```
*Pet.java  *Cat.java  *Dog.java  PetDriver.java

1
2 public class Dog extends Pet{
3     private int numBones;
4
5     public Dog(String oName, String pName, int bones){
6         super(oName, pName);
7         numBones = bones;
8     }
9
10    public void setNumBones(int bones){
11        numBones = bones;
12    }
13
14    public int getNumBones(){
15        return numBones;
16    }
17
18    public void speak(){
19        System.out.println("Woof");
20    }
21
22    public String toString(){
23        String output = super.toString();
24        output += new String(" and this dog has " +
25                            numBones + " bones.");
26        return output;
27    }
28 }
```



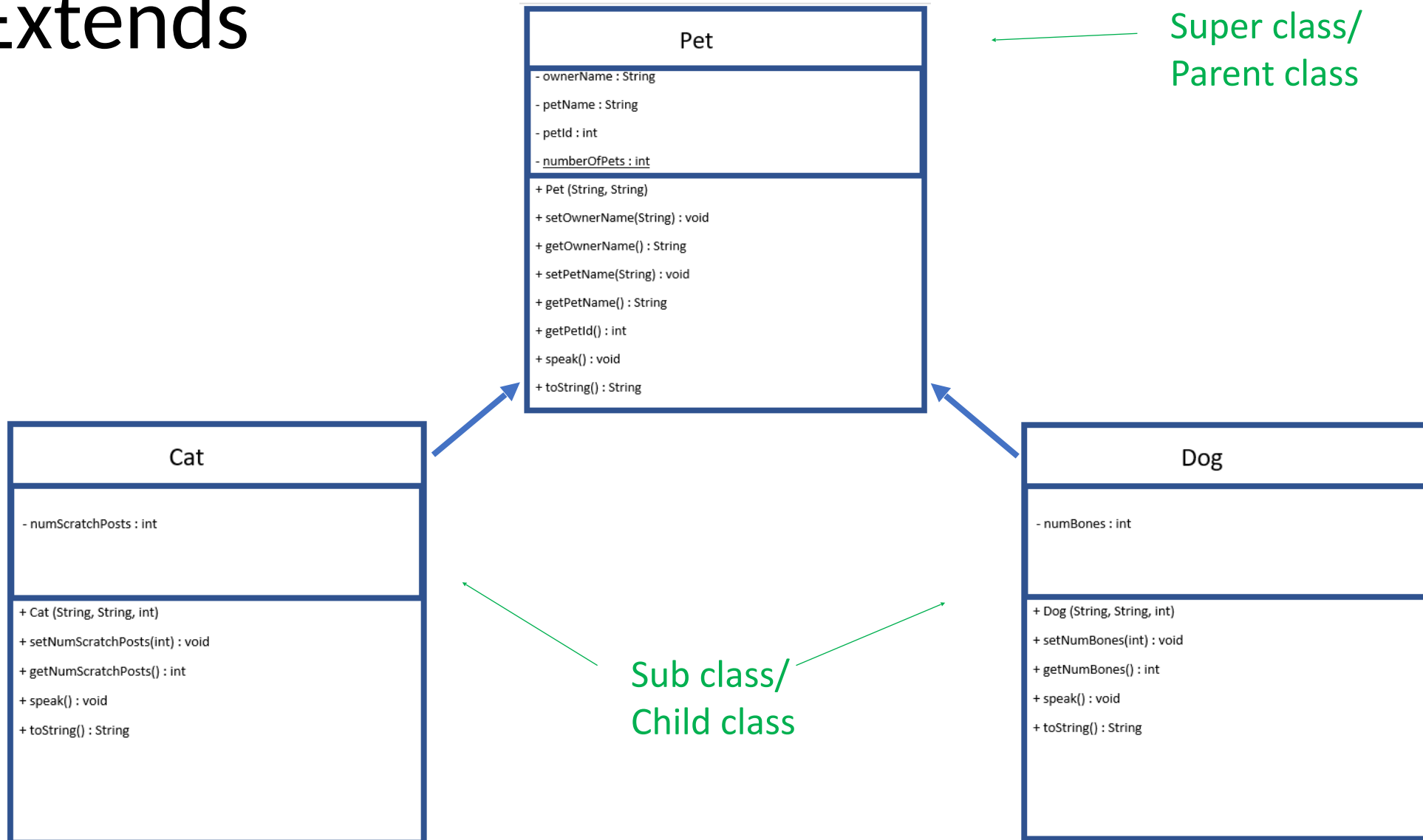
Cat

- numScratchPosts : int

+ Cat (String, String, int)
+ setNumScratchPosts(int) : void
+ getNumScratchPosts() : int
+ speak() : void
+ toString() : String

```
*Pet.java ✕ *Cat.java ✕ *Dog.java PetDriver.java
1
2 public class Cat extends Pet{
3     private int numScratchPosts;
4
5     public Cat(String oName, String pName, int posts){
6         super(oName, pName);
7         numScratchPosts = posts;
8     }
9
10    public void setNumScratchPosts(int posts){
11        numScratchPosts = posts;
12    }
13
14    public int getNumScratchPosts(){
15        return numScratchPosts;
16    }
17
18    public void speak(){
19        System.out.println("Meow");
20    }
21
22    public String toString(){
23        String output = super.toString();
24        output += new String(" and this cat has " +
25            numScratchPosts + " scratching posts.");
26        return output;
27    }
28
29 }
```


Extends





Try it!

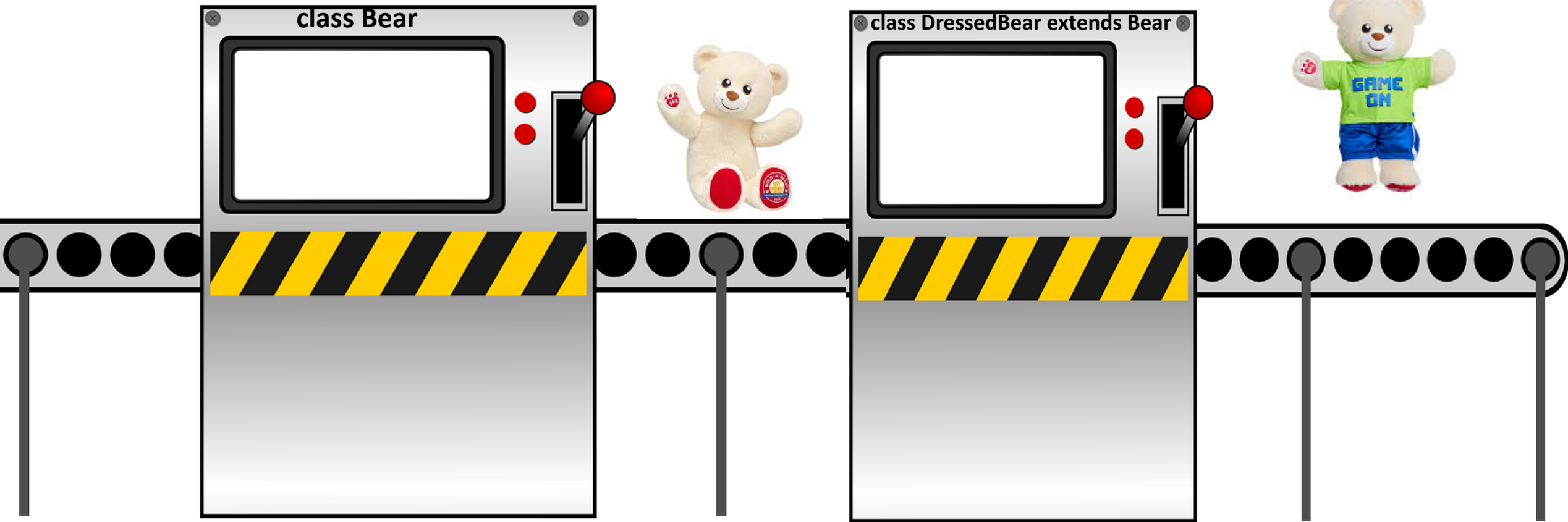
- In Eclipse, open the Dog Application
- Add an Animal class
- Move any instance variables that would describe any type of animal from the Dog class into the Animal class. Also move the getters and setters for those variables into the Animal class.
- Modify the Dog class header so that it extends the Animal class.
- Now fix the constructor(s). Pass any data that was used to initialize the variables that were moved to the Animal class by using the following format:

`super(data1, data2, ...);` *remember this has to be the first line of code inside the constructor.

This code is a call to a constructor in the parent class. Create a constructor in the Animal class that accepts all the information you passed and uses it to initialize the instance variables.

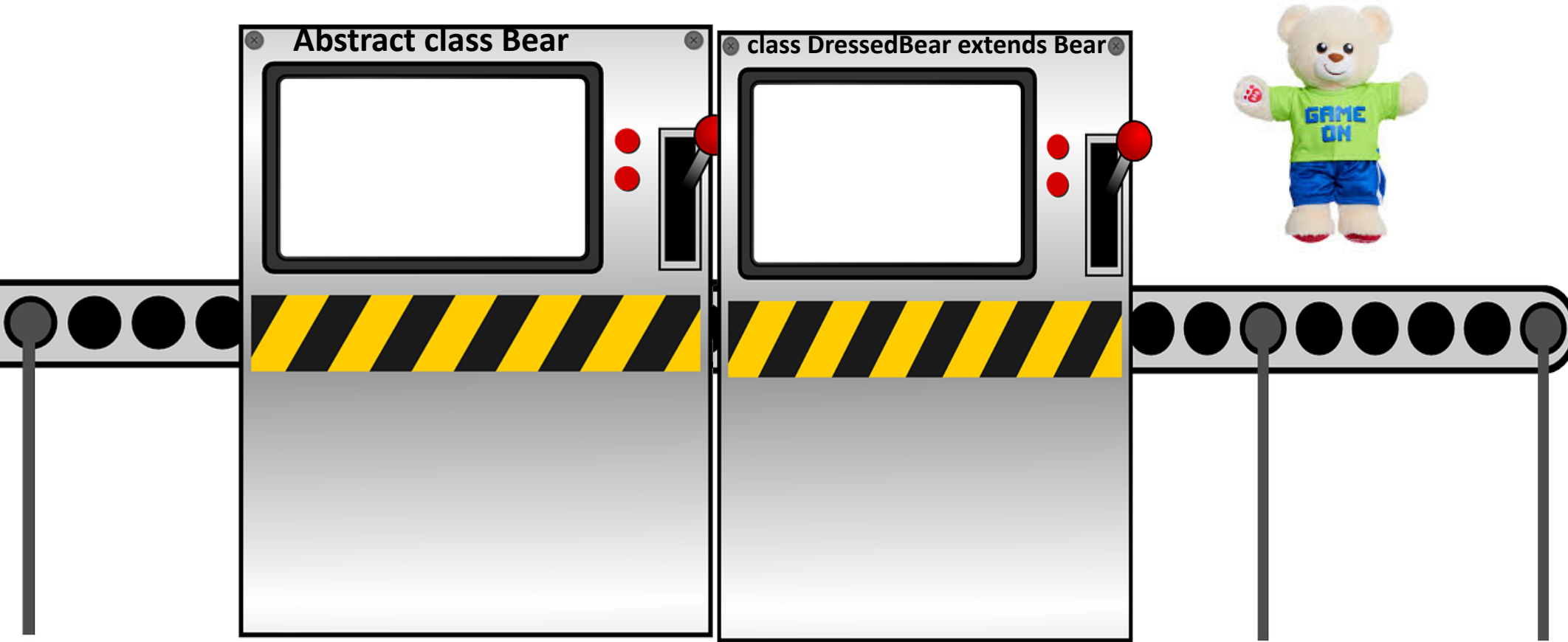
- Finally, fix the toString. Make a toString in the Animal class. Add a call to that toString in the Dog class. (See the dog or cat class on previous slides for hints).

Abstract Class





Abstract Class – can never be instantiated



Visibility Modifiers

Visibility Modifiers

Visibility	Public	Protected	Default	Private
From the same class	Yes	Yes	Yes	Yes
From any class in the same package	Yes	Yes	Yes	No
From a subclass in the same package	Yes	Yes (<i>Package, Inheritance</i>)	Yes (<i>Package</i>)	No
From a subclass outside the same package	Yes	Yes (<i>Inheritance</i>)	No	No
From any non-subclass class outside the package	Yes	No	No	No

Beware private variables of parent class!

Overriding methods

Revisit method signature and overloading

- Method signature includes method name, parameter type(s) – order matters. Return data type is NOT included in the method signature
 - float method1 (int number)
 - **int method1(int number)**
 - float method1(double number)
 - float method1(int number, double value)
 - float method1(double value, int number)

When an object calls a method

- Need car fixed
 - Ask Craig
 - `Craig.fixCar();`
 - If he doesn't know, he asks Uncle Don.
 - I didn't say `UncleDon.fixCar();`, even though that is what happens. I only ever asked Craig.
- Difference between overloading and overriding?



Try it!

- Create a method inside the Animal class called eat().
 - This method does not accept any parameters.
 - This method does not return anything.
 - This method will print the following phrase: “Yum, yum.”
- In the driver class, call the eat method using either yourDog or myDog.
- Did it work? Why?
- Create a method inside the Dog class called eat.
 - This method does not accept any parameters.
 - This method does not return anything.
 - This method will print “I am eating dog food.”
- Run your program again. What changed? Why?
- Now modify the eat method in the Dog class to first call the eat method of the Animal class with the following code:

```
super.eat();
```
- Run your program again. What changed? Why?

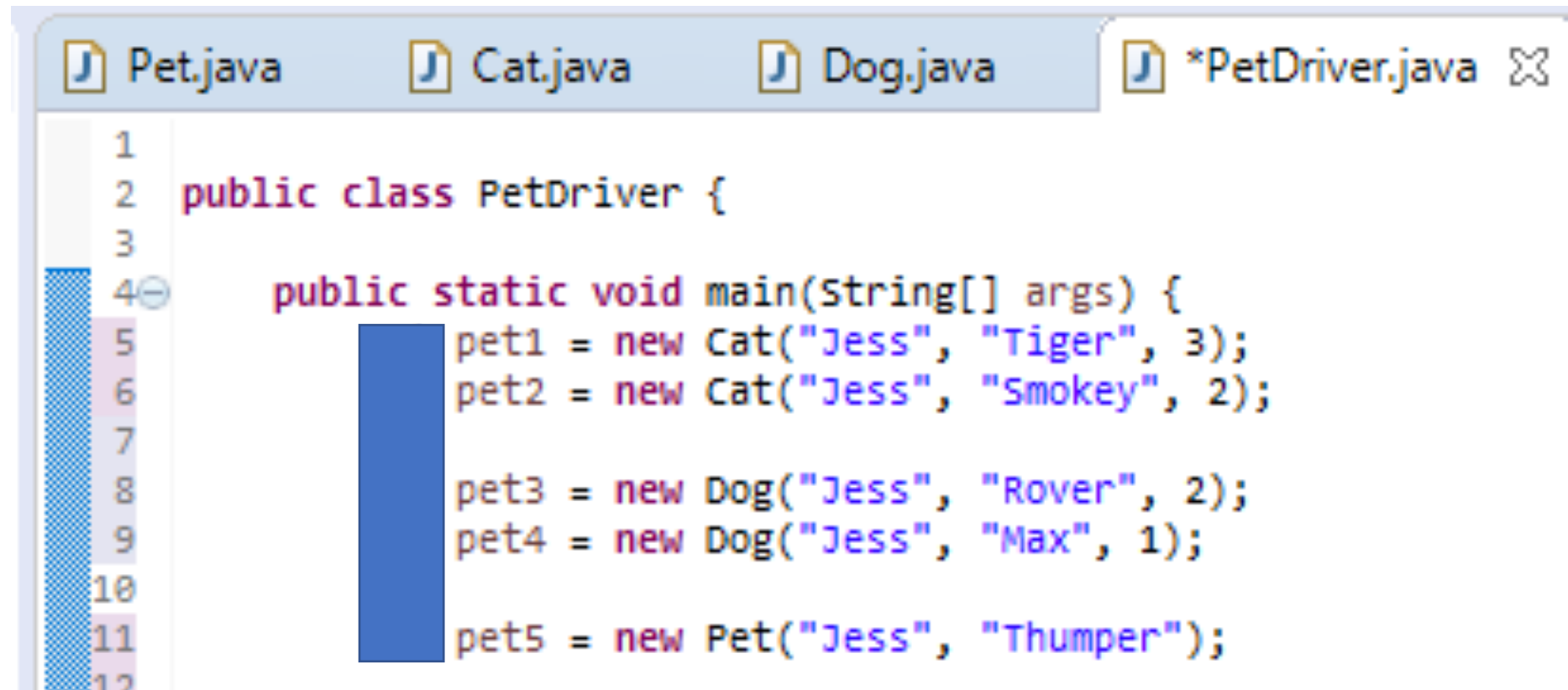
Polymorphism

Polymorphism

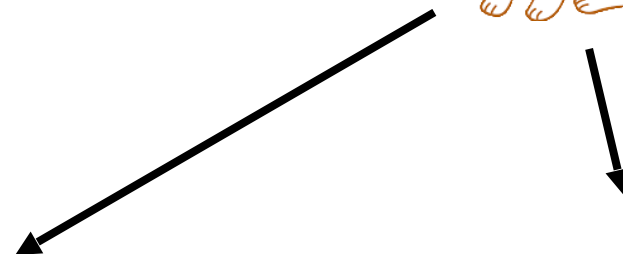
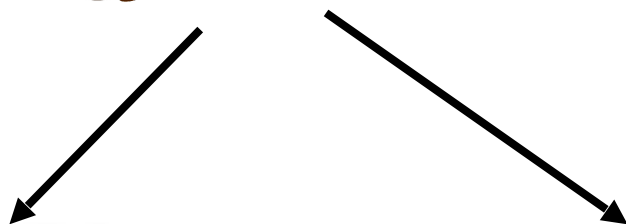
- Polymorphism is the idea that an object can be multiple things.
- I am:
 - Person
 - Mother
 - Daughter
 - Teacher

Polymorphism in Java

- Cat extends Pet
 - Pet1 is:
 - Cat
 - Pet
 - Pet2 is:
 - Cat
 - Pet
- Dog extends Pet
 - Pet3 is:
 - Dog
 - Pet
 - Pet4 is:
 - Dog
 - Pet
- Pet
 - Pet5 is:
 - Pet

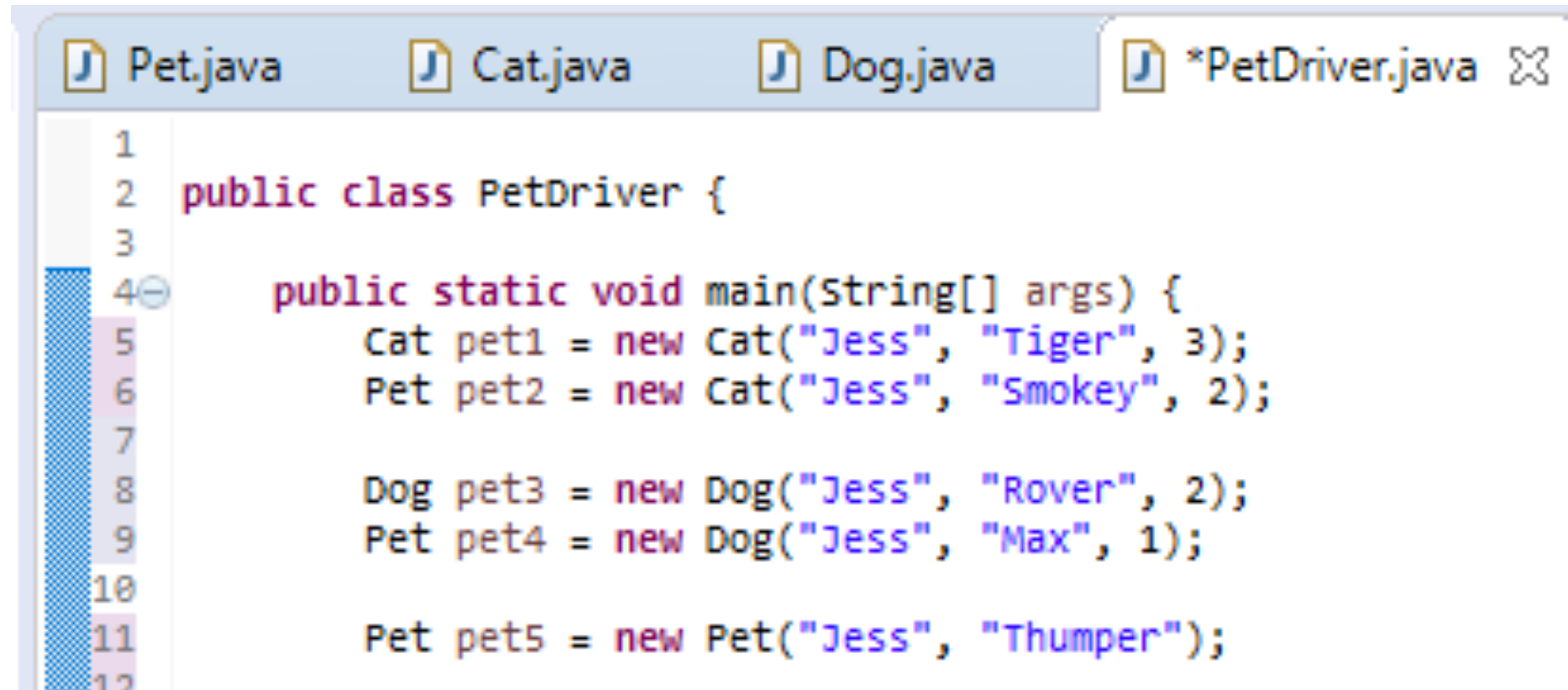


```
1
2 public class PetDriver {
3
4     public static void main(String[] args) {
5         pet1 = new Cat("Jess", "Tiger", 3);
6         pet2 = new Cat("Jess", "Smokey", 2);
7
8         pet3 = new Dog("Jess", "Rover", 2);
9         pet4 = new Dog("Jess", "Max", 1);
10
11         pet5 = new Pet("Jess", "Thumper");
12     }
```

Polymorphism in Java

- Cat extends Pet
 - Pet1 is:
 - Cat
 - Pet
 - Pet2 is:
 - Cat
 - Pet
- Dog extends Pet
 - Pet3 is:
 - Dog
 - Pet
 - Pet4 is:
 - Dog
 - Pet
- Pet
 - Pet5 is:
 - Pet



```
1
2 public class PetDriver {
3
4     public static void main(String[] args) {
5         Cat pet1 = new Cat("Jess", "Tiger", 3);
6         Pet pet2 = new Cat("Jess", "Smokey", 2);
7
8         Dog pet3 = new Dog("Jess", "Rover", 2);
9         Pet pet4 = new Dog("Jess", "Max", 1);
10
11         Pet pet5 = new Pet("Jess", "Thumper");
12     }
```

What if we want to switch containers?



Dog

```
5 Cat pet1 = new Cat("Jess", "Tiger", 3);
6 Pet pet2 = new Cat("Jess", "Smokey", 2);
7
8 Dog pet3 = new Dog("Jess", "Rover", 2);
9 Pet pet4 = new Dog("Jess", "Max", 1);
10
11 Pet pet5 = new Pet("Jess", "Thumper");
12
13 Pet pet1Forward = pet1;
14 Pet pet3Forward = pet3;
```



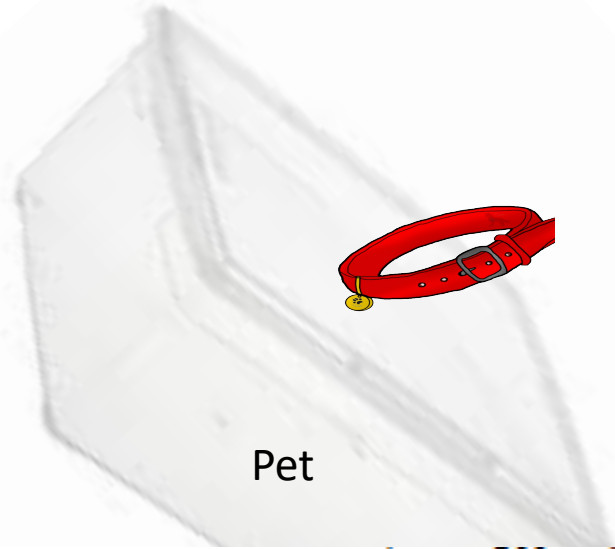
Pet



Cat



Pet



Pet



Dog



Cat

5
6
7
8
9
10
11
12
13
14

```
Cat pet1 = new Cat("Jess", "Tiger", 3);  
Dog pet3 = new Dog("Jess", "Rover", 2);  
  
Pet pet1Forward = pet1;  
Pet pet3Forward = pet3;  
  
Cat pet1Back = pet1Forward;  
Dog pet3Back = pet3Forward;
```



Pet



Pet

```
5 Cat pet1 = new Cat("Jess", "Tiger", 3);  
6  
7 Dog pet3 = new Dog("Jess", "Rover", 2);  
8  
9 Pet pet1Forward = pet1;  
10 Pet pet3Forward = pet3;  
11  
12 Cat pet1Back = (Cat) pet1Forward;  
13 Dog pet3Back = (Dog) pet3Forward;
```



Dog



Cat

* * * * Note * * * *

- The object didn't change. When a cat is forged, it will always be a cat.
- Can change what information is stored in the cat or dog
 - For example, can update number of bones for a dog
- Or can replace with a different object
- Not like Python!!! Not converting, just promising it is that type of object.

Try it!

- Decide if the following statements are valid or not? (In other words, will it compile?)

Pet copyPet = pet1;

Cat copyPet = pet2;

Pet copyPet = (Pet) pet1;

Cat copyPet = (Cat) pet2;

Pet copyPet = pet3;

Dog copyPet = pet4;

Dog copyPet = (Dog) pet5;

```
Cat pet1 = new Cat("Jess", "Tiger", 3);  
Pet pet2 = new Cat("Jess", "Smokey", 2);
```

```
Dog pet3 = new Dog("Jess", "Rover", 2);  
Pet pet4 = new Dog("Jess", "Max", 1);
```

```
Pet pet5 = new Pet("Jess", "Thumper");
```


Answer to try it:

```
Pet copyPet = pet1;
```

Valid – a cat can be stored in a pet bin, because a cat is a pet

```
Cat copyPet = pet2;
```

Invalid – Java does not know that the pet is a cat, because it is in the pet bin. So needs a promise that pet2 is really a cat.

```
Pet copyPet = (Pet) pet1;
```

Valid – but weird. Java knows a cat is a pet, so there is no need for the promise.

```
Cat copyPet = (Cat) pet2;
```

Valid – Java needs a promise that the thing stored in the pet bin is a cat.

```
Pet copyPet = pet3;
```

Valid – a dog can be stored in a pet bin, because a dog is a pet

```
Dog copyPet = pet4;
```

Invalid – Java does not know that the pet is a dog, because it is in the pet bin. So needs a promise that pet4 is really a dog. `Dog copyPet = (Dog) pet4;`

```
Dog copyPet = (Dog) pet5;
```

Invalid – there is no guarantee that pet5 is a dog, it could be any kind of pet. Be careful with your promises!!!

```
Cat pet1 = new Cat("Jess", "Tiger", 3);  
Pet pet2 = new Cat("Jess", "Smokey", 2);  
  
Dog pet3 = new Dog("Jess", "Rover", 2);  
Pet pet4 = new Dog("Jess", "Max", 1);  
  
Pet pet5 = new Pet("Jess", "Thumper");
```

instanceof

What if don't know what the object is?

```

1
2 public class LegoDriver {
3
4     public static void main(String[] args){
5         LegoSet set1 = new BrandedLegoSet();
6         LegoSet set2 = new LegoSet();
7
8         System.out.println("The set is a Lego set:");
9         System.out.println(set1 instanceof LegoSet);
10        System.out.println(set2 instanceof LegoSet);
11        System.out.println("\nThe set is a branded Lego set:");
12        System.out.println(set1 instanceof BrandedLegoSet);
13        System.out.println(set2 instanceof BrandedLegoSet);
14    }
15 }
16
17
18
19
20

```

<terminated> LegoDriver [Java Application] C:\Program Files\Java\jre1.8.0_231\bin\javaw.exe (May

The set is a Lego set:

true

true

The set is a branded Lego set:

true

false



Try it!

- In the AnimalShelter class, change the type of array to Animal rather than Dog. Modify the addAnimal method that accepts a whole Dog to accept an Animal instead.
- In the main method create a few animals of the Animal class and add them to the shelter.
- Create the following method in the AnimalShelter class.
 - countDogs(): Counts how many of the animals in the shelter are an instance of a Dog and returns that value.
- Call countDogs() on your shelter. Is it returning the correct result?