

Pet Shop

Your Tasks (Mark these off as you go)

- ☐ Define key vocabulary
- ☐ Interpret the relationship between super and sub classes
- ☐ Write the mealMaker classes
- ☐ Create objects using inheritance hierarchies
- ☐ Receive credit for this lab guide

☐ Define key vocabulary

Inheritance (as it applies to Java)

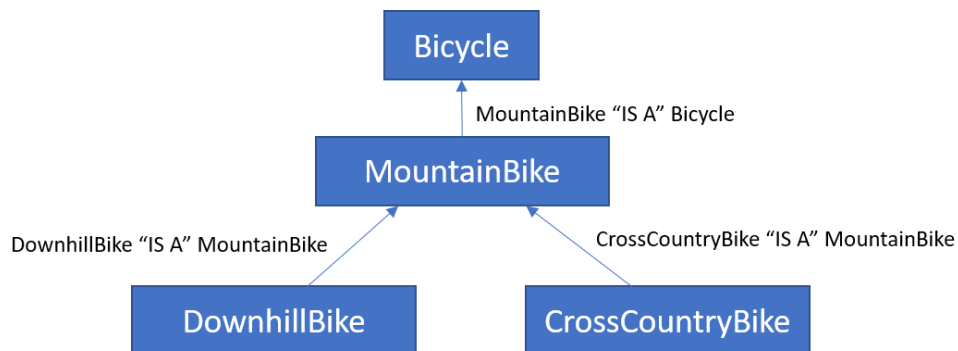
Super class

Sub class

☐ Interpret the relationship between super and sub classes

Recall that inheritance of one class from another follows an "IS A" relationship. That is, a mountain bike "IS A" bicycle. The reverse is not true, however. For example, a bike is not necessarily a mountain bike. When creating objects from super and sub classes, this relationship becomes important.

Consider the following hierarchy of inherited classes between bicycles




The above hierarchy shows the relationship among the classes in a program. According to the hierarchy

- CrossCountryBike "IS A" MountainBike
- DownhillBike "IS A" MountainBike
- MountainBike "IS A" Bicycle


When creating objects from super and sub classes this relationship is enforced. This is illustrated below,

MountainBike "IS A" Bicycle



```
Bicycle myBike = new MountainBike();
```

Bicycle is not necessarily a MountainBike



```
MountainBike myBike = new Bicycle();
```

Another way to think of this is that "parents can make children, but children cannot make parents"

Refer to the Pet, Cat, and Fish classes below,

```
public class Pet {

    private boolean vegetarian;
    private String type;
    private final int noOfLegs;

    public Pet(boolean vegetarian, String type, int noOfLegs){
        //sets the variables vegetarian, type, and noOfLegs declared
        //in this class to the parameters passed in the constructor
        this.vegetarian = veg;
        this.type = type;
        this.noOfLegs = legs;
    }

    public boolean getEats(){
        return vegetarian;
    }

    public int getLegs(){
        return noOfLegs;
    }

    Public int getType(){
        return type;
    }

    public String toString() {
        return "I am a : " + type;
    }

}
```

```
public class Cat extends Pet{

    public String name;

    public Cat(String name){
        super(false, "cat", 4);
        this.name = name;
    }

    public void speak(){
        System.out.println("Meow!");
    }

    public String toString() {
        return super.toString() + "\nMy name is : " + name;
    }
}

public class Fish extends Pet{

    public String name;
    public Fish(String n){
        super(false, "fish", 0);
        this.name = n;
    }

    public void speak(){
        System.out.println("Blub, Blub");
    }

    public String toString() {
        return "My name is " + name;
    }
}
```

(a) Draw a diagram to represent the relationship between the Pet, Cat, and Fish classes.

- (b) For each of the following (i) Indicate whether the statement is valid (V) or invalid (I) (ii) If the statement is not valid, indicate why.

Statement	V/I	If "I", indicate why.
Fish f = new Fish(false, "Fish", 0);		
Cat c = new Fish("Fred");		
Fish f = new Pet(true, "Fish", 0);		
Pet p = new Fish("Dori");		
Object o = new Cat("Fred");		
Object o = new Pet(317, 555, 1000);		

- (c) What is the value of s after the code block below? Do not include quotes in your answer.

```
Pet pet1 = new Pet(false, "Dog", 4);
String s = pet1.toString();
```

- (d) What is the value of s after the code block below? Do not include quotes in your answer. If an error occurs write "ERROR" AND indicate why the error occurs.

```
Cat pet2 = new Cat("Toby");
Pet pet3 = pet2;
pet3.speak();
System.out.print(pet2.toString());
```

- (e) Refer to the code block below to indicate what is printed for each of the following statements. If an error occurs write "ERROR" AND indicate why the error occurs.

```
Pet p = new Pet(true, "Spider", 8);
Cat c = new Cat("Roscoe");
Fish f = new Fish("Nemo");
```

- (i) `System.out.println(new Fish("Bob") instanceof Pet);` //instanceof returns true if Fish("Bob") instanceof Pet, otherwise it returns false

- (ii) `System.out.println(p);`

- (iii) `System.out.println(c);`

```

(iv) Pet[] myPets = new Pet[2];
    a. myPets[0] = p;
    b. myPets[1] = c;
    c. myPets[0].speak();

(v) f.speak();

(vi) System.out.println(f.getLegs());

(vii) System.out.println(c.getEats());

(viii) System.out.println(p.toString());

(ix) System.out.println(new Cat() instanceof Pet);

```

□ Write the meal maker classes

The Meal class and its subclass DeluxeMeal are used to represent meals at a restaurant. All Meal objects have the following attributes and methods.

- A String variable representing the name of the entree included in the meal
- A double variable representing the cost, in dollars, of the meal
- A toString method that indicates information about the meal

The following table shows the intended behavior of the Meal class.

Statement	Result
Meal burger = new Meal("hamburger", 7.99);	A new Meal object is created to represent a hamburger that costs \$7.99.
burger.toString();	The string "hamburger meal, \$7.99" is returned.

Write the complete Meal class. Your implementation must meet all specifications and conform to the behavior shown in the table above.

A deluxe meal, represented by a DeluxeMeal object, includes a side dish and a drink for an additional cost of \$3. The DeluxeMeal class is a subclass of Meal. The DeluxeMeal class contains two additional attributes not found in Meal:

- A String variable representing the name of the side dish included in the meal
- A String variable representing the name of the drink included in the meal

The following table shows the intended behavior of the DeluxeMeal class.

Statement	Result
<pre>DeluxeMeal burritoCombo = new DeluxeMeal("burrito", "chips", "lemonade", 7.49);</pre>	A new DeluxeMeal object is created to represent a deluxe meal including a burrito (entree), chips (side dish), and lemonade (drink). The cost of the burrito alone is \$7.49, so the cost of the meal is set to \$10.49.
<pre>burritoCombo.toString();</pre>	The string "deluxe burrito meal, \$10.49" is returned.

Write the complete DeluxeMeal class. Your implementation must meet all specifications and conform to the behavior shown in the table.

□ Create objects using inheritance hierarchies

The following Pet class is used to represent pets and print information about each pet. Each Pet object has attributes for the pet's name and species.

```
public class Pet
{
    private String name;
    private String species;

    public Pet(String n, String s)
    {
        name = n;
        species = s;
    }

    public String getName()
    {
        return name;
    }

    public void printPetInfo()
    {
        System.out.print(name + " is a " + species);
    }
}
```

The following Dog class is a subclass of the Pet class that has one additional attribute: a String variable named breed that is used to represent the breed of the dog. The Dog class also contains a printPetInfo method to print the name and breed of the dog.

```
public class Dog extends Pet
{
    private String breed;

    public Dog(String n, String b)
    {
        super(n, "dog");
        breed = b;
    }

    public void printPetInfo()
    {
        /* to be implemented */
    }
}
```

Consider the following code segment.

```
Dog fluffy = new Dog("Fluffy", "pomeranian");  
fluffy.printPetInfo();
```

The code segment is intended to print the following output.

Fluffy is a dog of breed Pomeranian

Complete the Dog method printPetInfo below. Your implementation should conform to the example above.

The PetMaker class contains the main method for the program. Write code that could be used to create the following pets,

- A rabbit named Floppy
- A dog (whose breed is pug) named Arty

The `PetOwner` class below is used to generate a description about a pet and its owner.

The `PetOwner` constructor takes a `Pet` object and a `String` value (representing the name of the pet's owner) as parameters.

```
public class PetOwner
{
    private Pet thePet;
    private String owner;

    public PetOwner(Pet p, String o)
    {
        thePet = p;
        owner = o;
    }

    public void printOwnerInfo()
    {
        /* to be implemented */
    }
}
```

Assume that `pet1` and `pet2` were created as specified above in the `PetMaker` class. The following table demonstrates the intended behavior of the `PetOwner` class using objects `pet1` and `pet2`.

Code Segment	Result Printed
<code>PetOwner owner1 = new PetOwner(pet1, "Jerry");</code> <code>owner1.printOwnerInfo();</code>	Floppy is a rabbit owned by Jerry
<code>PetOwner owner2 = new PetOwner(pet2, "Kris");</code> <code>owner2.printOwnerInfo();</code>	Arty is a dog of breed pug owned by Kris

Complete the `PetOwner` method `printOwnerInfo` below. Your implementation should conform to the examples. Assume that class `Dog` works as intended, regardless of what you wrote previously.

☐ **Receive credit for this lab guide**

Submit this portion of the lab to Pluska to receive credit for the lab guide. Once received, your completed code challenges will also be graded and will count towards your final lab grade.