Lab Polymorphism

CSIS-1410

Learning Objective:

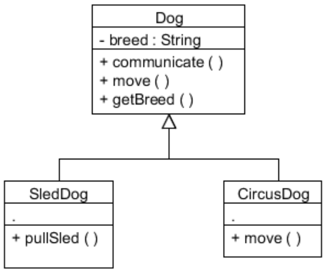
- Deepen your understanding of polymorphism
- Create an array of a superclass type
- Use the method **`getClass()`** to access the runtime class of the current object
- Practice the use of the **instanceof** operator

Instructions:

- Download the starter project from Canvas and unzip it in a folder with the same name. (If you change the folder name you will need to adjust the package name)
- Import the extracted (unzipped) code files into Eclipse.

  You can do that like this:
  - Right-click the src folder that should include the new package > **Import** .. the Import dialog opens
  - Select ***General*** > ***File System*** and click ***Next*** the Import from directory dialog opens
  - Use the Browse button to navigate to the folder **labInheritance** and click *OK*
  - Select the checkbox next to the folder **labInheritance**
  - IMPORTANT: Select the checkbox next to ***Create top-level folder***
  - Click ***Finish***
- Run App to make sure that the file import worked as expected.

  At this point the output should look like *Output 0*.



Output 0:

```
bark bark
run

bark bark
run

bark bark
tightrope walking
```

Output 1:

```
Dog: Greyhound
bark bark
run

SledDog: Husky
bark bark
run

CircusDog: Terrier
bark bark
tightrope walking
```

Make the following modification:

In class Dog:
Add an overridden toString method. Print the type of the class plus the breed separated with a colon and blank. (e.g. `Dog: Terrier`)
Here is how you can implement that functionality:
Use **getClass**, a method of java.lang.Object. It returns the runtime class of the object.
Then call the getSimpleName() method on the runtime class. Like this:

```
this.getClass().getSimpleName()
```

This will return the type of the class, in our case **Dog**.
Then use the plus operator to add a colon with blank and the breed.

In class DogApp:
Every time you create a new instance of a class add a statement that prints the newly created instance (see *Output 1*)
Hint: There is no need to call the toString method explicitly.
Compile and run. Your output should look like *Output 1* on the right.

Still in DogApp at the end of main do the following:
Print the header “Using an Array:”. This is to make the output more clear to the user.
Create an array of Dogs.
Use the array initializer to initialize the array with myDog, mySledDog, and myCircusDog
Use a foreach loop to loop through all the dogs
In the body of the foreach loop do two things:

1. print the current instance of the dog followed by a new line
2. call the method actAsDog and pass the current instance of the dog as argument

Compile and run. Now your output should look like Output 2 on the right

Still in DogApp do the following:
Inside the foreach loop right before the actAsDog method call check whether the current dog is-a SledDog.
You can do that by using the **`instanceof`** operator
E.g.:

```
if (object1 instanceof Type1) {
    //  do something
}
```

If the current dog happens to be a SledDog then call the method pullSled.
Hint:
In order to be able to access the method `pullSled` the Dog object still needs to be cast into a `SledDog` object. This cast is safe because we just checked the type of the Dog object with the `instanceof` operator.
Once we have a SledDog the method `pullSled` can be called.

Compile and run. Now your output should look like Output 3 on the right

Output 2:

```
Dog: Greyhound
bark bark
run

SledDog: Husky
bark bark
run

CircusDog: Terrier
bark bark
tightrope walking

Using an Array:

Dog: Greyhound
bark bark
run

SledDog: Husky
bark bark
run

CircusDog: Terrier
bark bark
tightrope walking
```

Output 3:

```
Dog: Greyhound
bark bark
run

SledDog: Husky
bark bark
run

CircusDog: Terrier
bark bark
tightrope walking

Using an Array:

Dog: Greyhound
bark bark
run

SledDog: Husky
pulling the sled
bark bark
run

CircusDog: Terrier
bark bark
tightrope walking
```