Week 2: Topics

- Objects
- Encapsulation

Creating an Object

- In Java, the **`new`** keyword is used to create objects

  An object is an instance of a class

- Three steps in creating an object

  1. Declaration
  2. Instantiation
  3. Initialization

Declaring a Class

Blackboard: Week2/Shapes/Rectangle.java

```
public class Rectangle
{
    private double length;
    private double breadth;

    public double getBreadth()
    {
        return breadth;
    }

    public void setBreadth(double rectBreadth)
    {
        breadth = rectBreadth;
    }

    public double getLength()
    {
        return length;
    }

    public void setLength(double rectLength)
    {
        length = rectLength;
    }
}
```

Creating an Object

Blackboard: Week2/Shapes/Shapes.java

```
public class Shapes
{
    public static void main(String[] args)
    {
        Rectangle myRectangle = new Rectangle();

        // Call to constructor
    }
}
```

Accessing Instance Variables and Methods

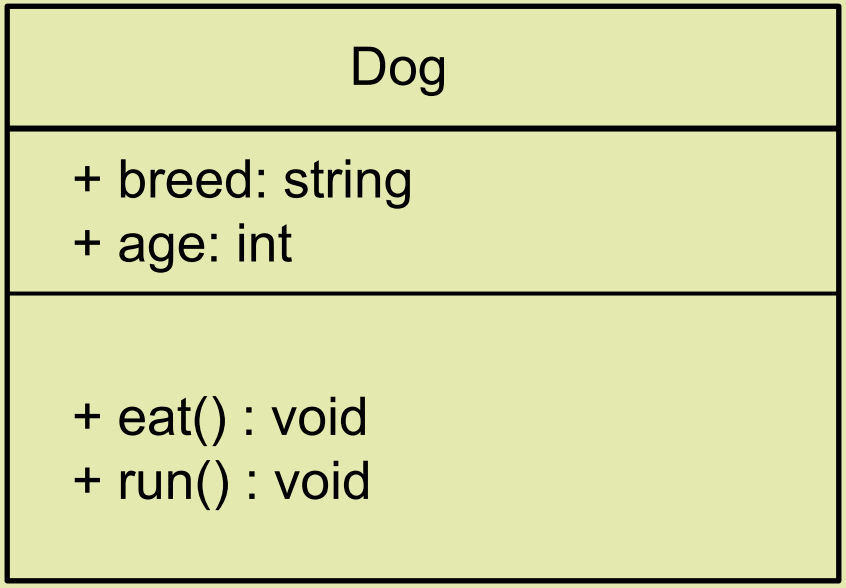
- Instance variables and methods are

  Accessed via created objects

- To access an instance variable
  - A fully qualified path should be used

```
Rectangle myRectangle = new Rectangle();
myRectangle.getLength();
```

Problem:



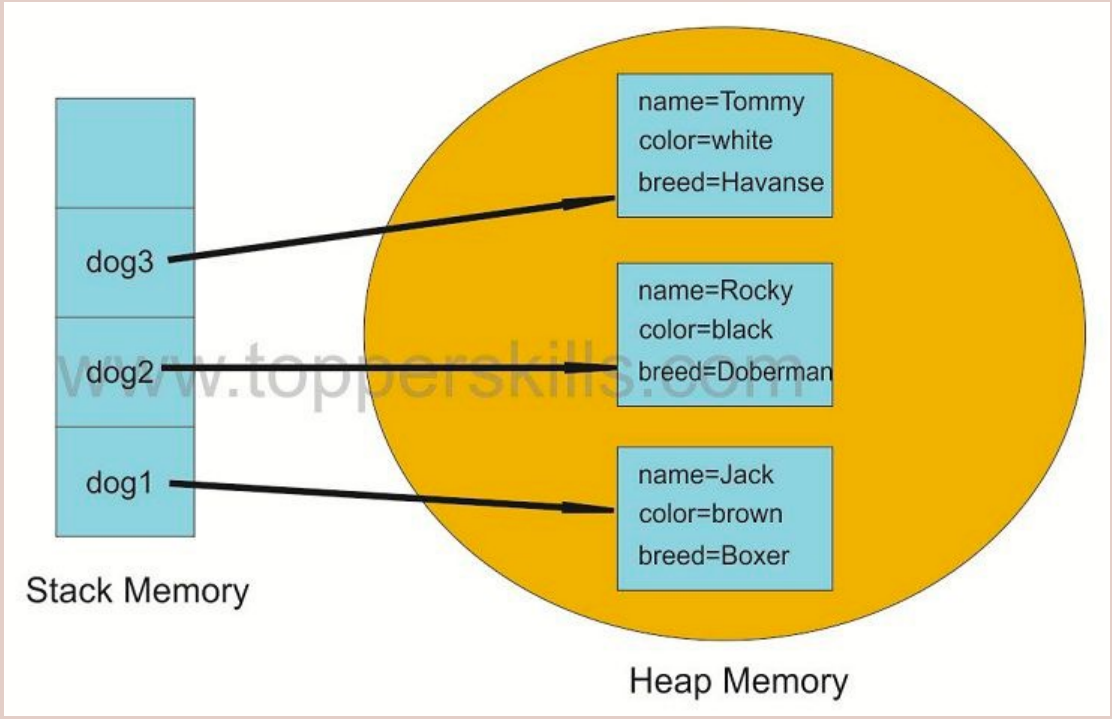
Explain the properties of the attributes and methods of the UML diagram.

Create objects of type Dog namely: `Mastiff` and `Maltese.`

Creating Objects

Blackboard: Week2/Dog

```
public class Dog
{
    public String breed;
    public int age;
    public void eat()
    {
 // eat something
    }
    public void run()
    {
 // run somewhere
    }
}
public class DogTest
{
    public static void main(String[] args)
    {
 Dog dog1 = new Dog();
 dog1.breed = "Mastiff";
 dog1.age = 2;
 Dog dog2 = new Dog();
 dog2.breed = "Maltese";
 dog2.age = 3;
    }
}
```



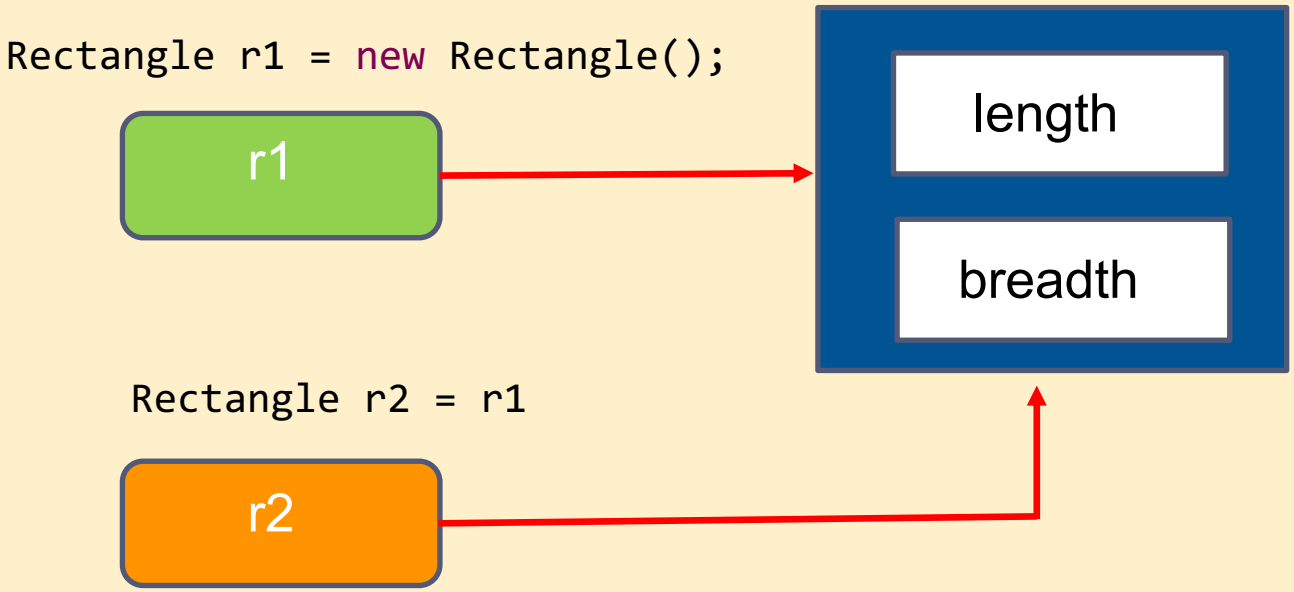
Stack vs Heap Memory

- Main differences between stack and heap memory are
  - Stack memory is used to store **local variables** and **function calls**
  - Stack memory is **contiguous**

Assigning Object Reference Variables

- A reference variable is used to *store the address* of the variable
  - We can assign the value of a reference variable to another reference variable
- Assigning an object reference variable does not
  - Create distinct object
  - Allocate memory
  - Create duplicate copies

Assigning Object Reference Variables



All reference variables refer to the same object

Assigning Object Reference Variables

- A reference **always** has a valid address **or** possibly a *special value* to signify that it is invalid

  In Java, the special value is signified using the keyword null

```
MyClass m = null;   // Not initialised
…
m = new MyClass();
```

A quick tutorial:

```
Rectangle r1 = new Rectangle();
Rectangle r2 = r1
…
r1 = null;
```

What happens to r2?

Instance Methods

- An instance method is a method that acts upon an instance variable
- To call an instance method
  - We create an object of the class within which it is defined
- There are two types of instance methods

  Accessor methods

  Mutator methods

Accessor Method

- These methods access or read instance variables
  - Do not modify the instance variable

```
public class Rectangle
{
    private double length;
    private double breadth;

    public double getBreadth()
    {
        return breadth;
    }

    public double getLength()
    {
        return length;
    }
}
```

Mutator Method

- These methods modify the instance variables

```
public class Rectangle
{
    private double length;
    private double breadth;

    public void setBreadth(double rectBreadth)
    {
        breadth = rectBreadth;
    }

    public void setLength(double rectLength)
    {
        length = rectLength;
    }
}
```

Problem:

Create a `RectangleTest` class which instantiates a `Rectangle` in the `main()` method, and assigns a value to the variable `length`.

What is the problem?

Blackboard: Week2/Rectangle/RectangleTest.java

Encapsulation

- Another name for encapsulation is

  Information Hiding

- The basic idea is that a class should expose a clean interface

  But nothing about its internal state

- We can change the internal implementation whenever we like

  So long as we do not change the interface

Encapsulation

Blackboard: Week2/Student1/Student.java

Problem:

Create a `Student` class with a private `age` attribute and with its own mutator and accessor methods.

Write a mainline to create an object of type `Student` and call its mutator method.

Encapsulation

Blackboard: Week2/Student2/Student.java

Encapsulation

Blackboard: Week2/Student2/StudentTest.java

Creating/Storing a Collection of Objects

- What if we needed to create and save a number of bank customers?

  We could use an array

    Remember that an array is a collection of similar types

Problem:

Create a `Rectangle` class and declare an array of rectangles.

Blackboard: Week2/RectangleArray