Class 1

Object Oriented Programming

The three basic principles of object oriented programming are

1. Encapsulation
2. Inheritance
3. Polymorphism

Encapsulation –

Has to do with *objects* and *classes*

An ***object*** is an entity that has

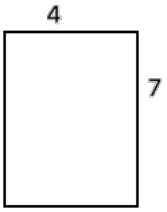
1. Attributes (data)
2. Behaviors (methods)

Example

A TV's remote control is an object

volume ^ volume v channel ^ channel v on/off	**The data or attributes:** channel (int) volume (int) onOff (boolean) **The behaviors or methods** volUp() volDown() chUp() chDown() onOff()

The ***state*** of a remote object might be: channel = 4 volume = 12 on/off = true (for on)	channel 4 volume 12 on/off true
The ***state*** of a different remote object might be: channel = 32 volume = 1 on/off = true (for on)	channel 32 volume 1 on/off true

A **rectangle** is an object.
 	
Its **attributes (data)** might be
length (int)
width (int)
 	
Its **behaviors (methods**) might be
int area() // returns the area
int perimeter() // returns the perimeter
int getLength() // returns the length
int getWidth() // returns the width
void setLength (int len) // sets the length
void setWidth (int wid) // sets the width
	A typical rectangle object
	 
	
	Here the attributes/data are
	length =7; width = 4
	
	The method area() returns 28
	The method perimeter() returns 22

A **Dog** is an object
 	
Attributes or data
name (String)
breed (String)
Weight (int)
Behaviors or methods
String getName() // returns the name
String getBreed() // returns the breed
int getWeight () // returns the weight
void setName(String n)
void setBreed(String b)
void setWeight(int w)
	Here is one Dog object
	 
	Attributes/ data
	name = “Brian”
	breed = “Labrador”
	weight =60
	
	The method getName() returns “Brian”
	etc
		Here is another Dog object
		 
		Attributes/ data
		name = “Scooby-Doo”
		breed= “Great Dane”
		weight = 150

Different Dog objects have different attributes/data but all share the same behaviors/methods

So where do classes fit into the picture?

A **class** is a template or blueprint for an object.
A class specifies the attributes and behaviors of a typical object

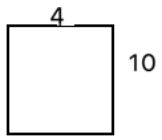
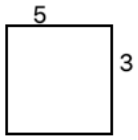
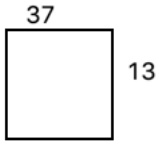
Just as a builder can take a blueprint from an architect and build many houses from the blueprint, a programmer can create many objects from a class.

Example (This should be somewhat of a review from last semester)

```
public class Rectangle       //the name of a class should begin with an upper case letter
{
     private int length;        // Don't worry about the word "private" for now
     private int width;        // These are the attributes or class variables,

    // here are the methods or behaviors
     public  int area()  // the methods can access the class variable directly
     {
        return length * width;
     }

    public int perimeter()
    {
         return 2*(length + width);
    }

    public  int getLength()
    {
         return length;
    }
    public  int getWidth()
    {
         return width;
    }

    public void setLength(int len)
    {
            length = len;
    }
   public void setWidth(int wid)
   {
         width = wid;
    }
}
```

This class describes all Rectangle objects. It is a blueprint of template for creating Rectangles. (If you do not know what private and public mean, we will discuss that later)

Once we have the blueprint for Rectangle objects we can create or **INSTANTIATE** any number of Rectangles using the "**new**" operator

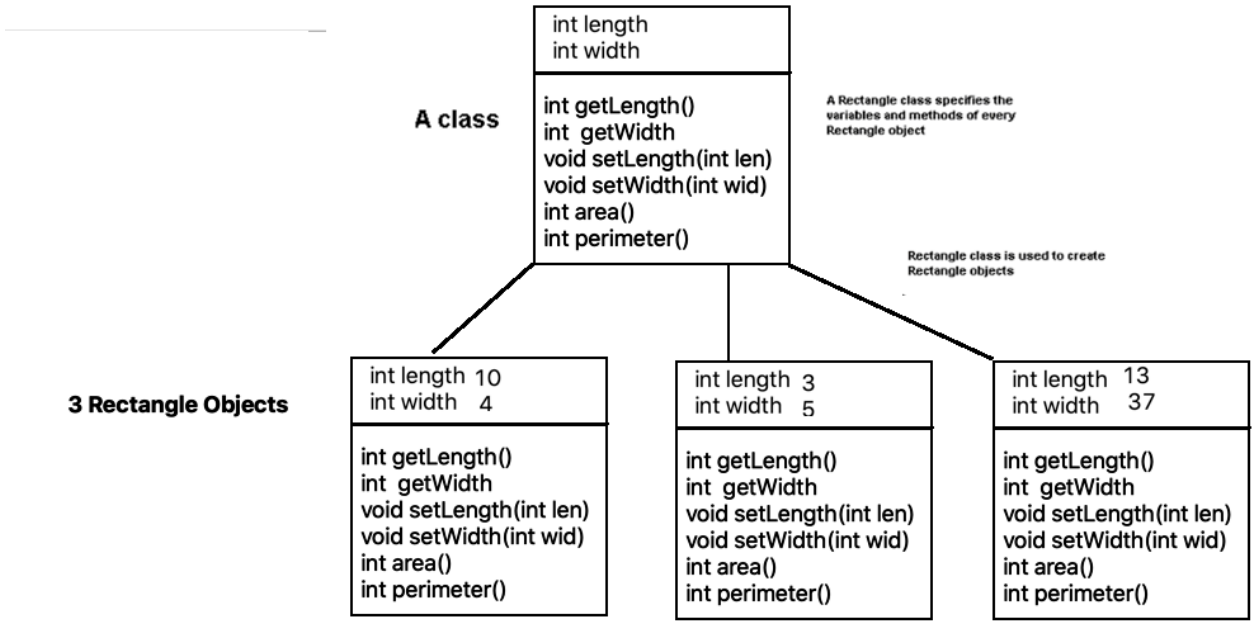
Rectangle **a = new Rectangle ();**
a.setLength(10);
a.setWidth(4);
 
	Rectangle **b = new Rectangle ();**
	b.setLength(3);
	b.setWidth(5);
	 
		Rectangle **c = new Rectangle ();**
		c.setLength(13);
		c.setWidth(37);
		 

System.out.println(a.area()) prints 40 // notice the **.** when invoking a method

System.out.println(b.area()) prints 15

System.out.println(c.area()) prints 481

Here is another way to picture the Rectangle class and Rectangle objects

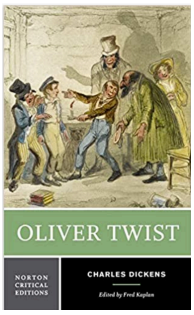
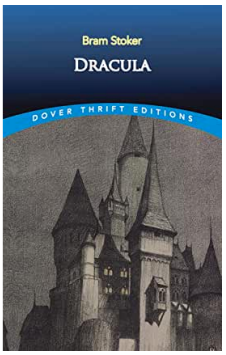


Here are three Rectangle objects a, b, and c created with "new"

So what is *encapsulation*?

Encapsuation is the language feature that packages attributes (data) and behaviors (methods) into a single unit called an object. That is, data and methods are bundled together in a single object.

Example:

public class Book
{
private String title;
private String author;
private double price;
public void setTitle(String t)
{
title = t;
}
public void setAuthor(String a)
{
author = a ;
}
public void setPrice(double p)
{
Price = p ;
}
}
	Create Book objects
	
	Book b = **new** Book();
	b.setTitle("Oliver Twist");
	b.setAuthor("Dickens")
	b.setPrice(19.38);
	
	 
	
		Book c = **new** Book();
		c.setTitle("Dracula");
		c.setAuthor("Stoker")
		c.setPrice(6.98);
		
		 

Before we write our own classes, we will look at some of the classes that Java provides. Java provides many (thousands) of classes. Just as I created the Rectangle, Dog, and Book classes, Java provides many blueprints that you can use to create or instantiate objects

One last detail:

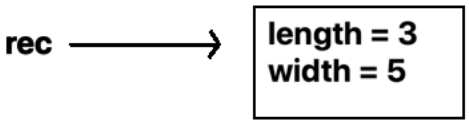
You create an object from a class using the "new" operator. For example

Rectangle rec = new Rectangle(3,5);

The data for the object is stored in the computer's memory at some unspecified memory location/address.

The variable **rec** holds that address. The variable rec is called a reference or a pointer.

This is often pictured as:



One class that you have already seen is the **String** class.
We will review the Sting class.

With the Sting class Java is the architect. Java provides the blueprint. You are the builder. You use Java's blueprint to create any number of string objects.

What are the attributes/data of a String object? The data is a sequence of characters.
Here we create a String object:

String geek = new String ("Sheldon");

We created a String object and its data is "Sheldon"

What are the methods? There are dozens. Some that you have seen are:

```
int getLength()        // returns the number of charac
char charAt(int n)     // returns the character at position n, the first character is at position 0
String substring(int a,int b)
                       // returns the characters from position a up to but not including pos b
```

So geek.getLength() returns 7 // "Sheldon has 7 characters
geek.charAt(4) returns 'd' // 'd' is in position 4, remember the first position is 0
Geek.substring(2,6) returns "eldo" // characters in positions 2 through 5

References:

The statement

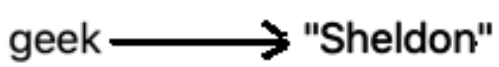
```
String geek = new String("Sheldon");
```

Creates a String object that is stored in the computer's memory

As with the Rectangle example, a memory location or address is stored in geek.
In other words, geek holds the memory address where "Sheldon" is stored.

- geek is called a **reference** or pointer.
- We say that geek is a reference to "Sheldon" or geek points to "Sheldon"
- ***A reference*** is a memory location.

We can picture this as



Example:

```
public class Example
{
   public static void main(String[] args
   {
         String  clown = new String("Krusty");  // clown is a reference to "Krusty"
         System.out.println(clown);              // prints the characters
         System.out.println(clown.charAt(0));
         Systen.out.println(clown.length());
         System.out.println(clown.substring(0,5);
  }
}
```

The statement

```
         String  clown = new String("Krusty");
```

Creates a String and stores its address in the variable clown

Output is
Krusty
K
6
Krust

A shortcut: You can create a string object without the keyword new:
String geek = "Sheldon"; // rather than geek = new String("Sheldon");
String clown = "Krusty";

Another String operation that you saw last semester is ***concatenation***
Concatenation is the joining of strings

```
         String s = "Sheldon"
         String c = "Cooper"
         String name = s+"  "+ t;   // join s , a blank,  and t into a new string
         //name is the String "Sheldon Cooper"; notice that I included  a blank
```

```
String k = "Krusty";
String c = "Clown";
String fullName = k + " the " + c;
```

fullName is the string "Krusty the Clown"

Important string property: Strings are immutable

Once a String is created, it cannot be changed.

Example

String s = “Sheldon”;	s → "Sheldon"
s = s + “ Cooper”	"Sheldon" s → "Sheldon Cooper"

Concatenation creates a new String. Concatenation copies “Sheldon” into a new String and adds “ Cooper” to that.

Example:

String s = ‘a’	s → “a”
s = s + “b”;	“a” s → “ab”
s = s + “c”;	“a” “ab” s →”abc”
s = s + “d”;	“a” “ab” “abc” s →”abcd”
s = s + “e”;	“a” “ab” “abc” “abcd” s →”abcde”

Each time a letter was added to the string a new string was created.
So if s is currently “abcd” then
S = s+ “e”

- Makes a copy of “abcd”
- Adds “e” making the string “abcde”
- Assigns the new string to s

Strings are immutable. You cannot alter a string. If you want to add “e” Java makes a new string.

The original string (“abcd”) is now longer accessable and is called an orphan.

Another String method is toUpperCase()

Example

String s = “abc”	s → “abc”
s = s.toUpperCase()	“abc” s → “ABC”

The method toUpperCase() does NOT change the original string “abc” to “ABC”
STRINGS ARE IMMUTABLE
Instead it creates a new String (“ABC”) and assigns “ABC” to s.
The original string (“abc”) is no longer referenced by any variable. It is inaccessible.
It is called an *orphan.*

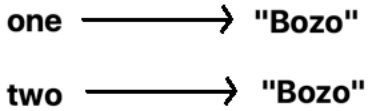
How to test equality of strings.

Assume that you create two String objects:
String one = new String(“Bozo”);
String two = new String(“Bozo”);

What is the value of the expression

one == two

Is it true or false???
One is holding the address of the first String and two is holding the address of the second String



Even though the characters are the same one and two store two different addresses.
The expression one == two compares the ADDRESSES stored in one and two. They are different so the expression one == two is false.
With Strings == compares references (addresses in memory)

To determine whether two strings are identical character by character use the equals(…) method

one.equals(two)

This will return true. Both strings have identical characters (in the same order).
Basic Rule:

USE equals(…) WHEN COMPARING STRINGS

There are very few cases when you want to compare addresses and use ==

Here are a few methods of the String class – there are many more

Some String methods

Method	Explanation	Example
char charAt(int index)	s.charAt(i) returns the character at index *i*. All Strings are indexed from 0.	String s = "Titanic"; s.charAt(3) returns 'a' (indexing starts at 0)
int compareTo(String t) **DO NOT USE > or < TO COMPARE TWO STRINGS**	compares two Strings, character by character, using the ASCII values of the characters. s.compareTo(t) returns a negative number if s < t. s.compareTo(s) returns 0 if s == t. s.compareTo(t) returns a positive number if s > t.	String s = "Shrek"; String t = "Star Wars"; s.compareTo(t) returns a negative number. s.compareTo(s) returns 0. t.compareTo(s) returns a positive number
int compareToIgnoreCase(String t)	similar to compareTo(...) but ignores differences in case.	String s = "E.T."; String t = "e.t."; s.compareToIgnorecase(t) returns 0.
boolean equals(Object t) (The strange parameter will make sense later. For now, think of the parameter as String)	s.equals(t) returns true if s and t are identical character by character.	String s = "Star Trek"; String t = "STAR TREK"; s.equals(t) returns false s.equals("Star Trek") returns true
boolean equalsIgnoreCase(String t)	s.equalsIgnoreCase(t) returns true if s and t are identical, ignoring case.	String s = "STAR TREK"; String t = "Star Trek"; s.equalsIgnorecase(t) returns true
int indexOf(String t)	s.indexOf(t) returns the index in s of the first occurrence of t and returns -1 if t is not a substring of s.	String s = "**The** Lord Of The Rings"; s.indexOf("The") returns 0; s.indexOf("Bilbo") returns –1.
int indexOf(String t, int from)	s.indexOf(t, from) returns the index in s of the first occurrence of t beginning at index from; an unsuccessful search returns –1.	String s = "The Lord Of **The** Rings"; s.indexOf("The", 6) returns 12;
int length( )	s.length() returns the number of characters in s.	String s = "Jaws"; s.length() returns 4
String replace(char oldChar, char newChar)	s.replace(oldCh, newCh) returns a String obtained by replacing every occurrance of oldCh with newCh.	String s = "Harry Potter"; s.replace ('r','m') returns "Hammy Pottem"

String substring(int index)	s.substring(index) returns the substring of s consisting of all characters with index greater than or equal to index.	String s = “The Sixth Sense”; s.substring(7) returns “th Sense”
String substring(int start, int end)	s.substring(start, end) returns the substring of s consisting of all characters with index greater than or equal to start and strictly less than end.	String s = “The Sixth Sense”; s.substring(7, 12) returns “th Se”
String toLowerCase()	s.toLowerCase() returns a String formed from s by replacing all upper case characters with lower case characters.	String s = “The Lion King”; s.toLowerCase() returns “the lion king”
String toUpperCase()	s.toUpperCase() returns a String formed from s by replacing all lower case characters with upper case characters.	String s = “The Lion King”; s. toUpperCase() returns “THE LION KING”
String trim()	s.trim() returns the String with all leading and trailing white space removed.	String s = “ Attack of the Killer Tomatoes “; s.trim() returns “Attack of the Killer Tomatoes”