ES 1036 Programming Fundamentals

Class and Object

> *"There are no secrets to success. It is the result of preparation, hard work, and learning from failure"* ~Colin Powell

Dr. Quazi M. Rahman, PhD, PEng, SMIEEE
Office Location: TEB 361
Email: QRAHMAN@eng.uwo.ca
Phone: 519-661-2111 x81399

> *"Genius is 1% talent and 99% percent **hard work**..."* ~ Albert Einstein

Outline

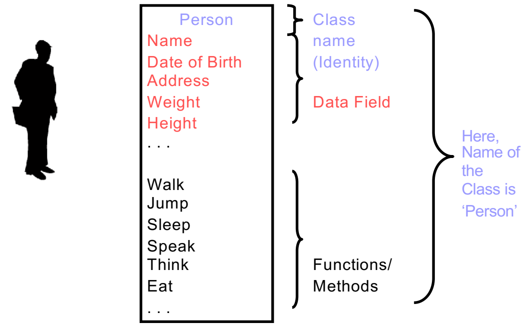
- Class and Object
- Designing a Class
  - □Class declaration
  - □Class implementation
  - □Using classes
- Constructors
- Accessor/mutator
- Program examples
- Object oriented programming (OOP)
- Principles of OOP

Classes

- A ***class*** is a program-structure / blueprint that allows a programmer to define new data types by following the object oriented programming principles.
- A ***class*** can be used
  - □ to add functionality to an existing data type or
  - □ to create a new data type.
- A ***class*** definition combines data and functionality/behavior.
- A Java class uses variables to define data fields and methods to define behaviors.
  - □ Additionally, a class provides a special type of method, known as constructor, which are invoked to construct objects from the class.

A ***class*** is a program-structure that allows a programmer to define new data types

Class: an analogy



Person
Name
Date of Birth
Address
Weight
Height
. . .

Walk
Jump
Sleep
Speak
Think
Eat
. . .

Class name (Identity)

Data Field

Functions/ Methods

Here, Name of the Class is 'Person'

Primitive Data Types and Class: An Analogy

Class declaration syntax

- Syntax of a Class: (We already have this idea)
  access modifier label class any_valid_name {
  /* data-field/data-members and methods /function-members are declared here. We do not initialize any data member here (except for the static final data members)*/
  access modifier label data member/field/ instance variables declarations;
  access modifier label method definition;
  }
- Access **modifier** labels for any outer class can be public, abstract and final. Only inner class can have private access modifier.
  - □ When a class is declared as a separate entity it is called an outer class.
  - □ When a class in declared inside another class, we call it an inner class.
- Access **modifier** labels for any field or method can be public, private and protected.
- If there is no access modifier used, it is considered to have default modifier. (there is a fine difference between default and public modiier)
- **Note: We can not use any access modifier for the class if that class is declared inside the same file in a package where there already exists another class.**
- *To make sure we can play with access modifiers, it is good to create a second file for a second class in the same package (see the lab handout). Also, we can create different packages to contain different classes.*

Class declaration Example

- Example: Below, three different data-fields / data-members (string, int and float type) and a method (printInfo()) / member-method are collected together to form a new data-type / blue-print called Student.

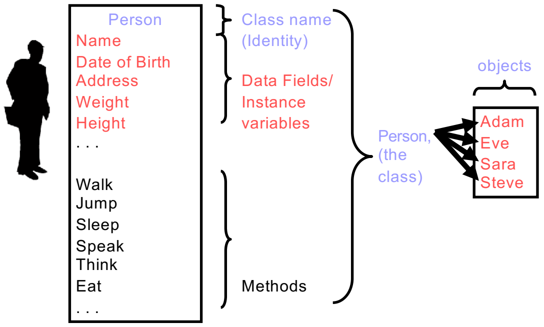
```
class Student{
 String name;
 int ID;
 float score;
 void printInfo(){
   System.out.println(name+" (ID:
   "+ID+") scored "+ score );
 }
}
```

Object

- An ***object*** is a reference-variable (in Java) of a defined ***class*** type, also referred to as an ***instance** of a **class***. Since an object is an instance of a class, the data members/fields are also known as instance variables.
- Example: If ‘Student’ is a class (a new data-type) then reference objects can be declared as:
  - □ Student Adam;
  - □ Student Eve;
- In the above example, the objects Eve and Adam refer to all the characteristics (data and functionalities) declared in the class Student, which can be accessed via member access operator (dot operator).

An *object* is a *class* type variable

Class and object: an analogy



Person
Name
Date of Birth
Address
Weight
Height
. . .
Walk
Jump
Sleep
Speak
Think
Eat
. . .

Class name (Identity)

Data Fields/ Instance variables

Methods

Person, (the class)

objects

Adam
Eve
Sara
Steve

Class-Name and objects ≈ Data-type and variable-name

- The relationship between class-name and object is equivalent to the relationship between data-type and variable-name.
- Declaring an object of `Student` (here, `Student` is a class name) type data follows similar approach of declaring a variable of `int` type data as shown below:
  - □ `Student x;` //**x** is the name of a reference object of **Student** type
  - □ `int y;` // **y** is a name of a variable of **int** type
- Important Note: Objects are always reference-objects in Java that require new spaces in the heap to store the associated data-fields.
  - □ `Student x;` /*declare an object in the stack that refers to **null**; points to nowhere.*/

    `x = new Student();` /*x is now pointing to the first location in the heap that contains x’s data field(s).*/ OR together,
  - □ `Student x = new Student();`

class in a Program

```
public class MyClass{
 public static void main(String[] args){
  Scanner input = new Scanner(System.in);
  //declaring two Student type objects Q and R
  Student Q = new Student(), R = new Student();
  //assigning values to all the data-members/fields for Q
  Q.score = 100;
  Q.ID=1111;
  Q.name="Eric";
  System.out.print("Enter R's Name: "); R.name = input.next();
  System.out.print("Enter R's score: "); R.score = input.nextFloat();
  System.out.print("Enter R's ID: "); R.ID = input.nextInt();
  //calling member method printInfo()
  Q.printInfo(); R.printInfo();
 }
}
//In the same file, a new class called Student is defined below
class Student{
 String name;//Data-field/Data-member name
 int ID; //Data-field/Data-member ID
 float score; //Data-field/Data-member score
 void printInfo(){//Method/Function-member printInfo()
  System.out.println(name + " (ID: "+ID+") scored "+ score );
 }
}
```

Accessing data members of a Class

- After an object (variable) of *Student* type data is declared, its data members can be accessed using the dot operator (.), also known as the *member access operator*.
- Member access operator is a period placed between the reference object’s name (here, Q and R are the object names) and a member name (e.g., ID, name, score).
- Example: Q.ID refers to the data member ID for the object Q.

```
/*Example on Member access
  operator*/

Student Q = new Student();
Q.score = 100;
Q.ID=1111;
Q.name="Eric";
```

Why Class?

- Class provides the option to store collections of related data items (and functions) that may have the same or different data types.
- Difference between Class and array: arrays are useful data structures for storing a collection of data elements of the same data type.
- Class supports repetitive structure that contains different data-fields and methods.

Initializing objects using `Constructors`

- The objects in a program are **initialized** using a special method, called constructor; the initialization is done inside the definition of the constructor method.
- This special public member method (can be private too in special scenario) must have the same name as the class itself.
- Constructors do not have a return type, not even void.
- A constructor method is automatically called /invoked when the object reference constructs an object in the heap with the initialized values assigned to its members by referring those using the new operator.
- Like any other method, constructors can be overloaded (i.e., multiple methods with the same name but different parameter list), making it easy to construct objects with different initial data values.

Example: Student class using overloaded constructors

```
public class Student {
 String name;
 int ID;
 double score;
 Student(){
  name = "Rebecca"; ID = 111; score =100;
 }
 Student(String n){
  name = n; ID = 999; score = 92;
 }
 Student (int i, double s){
  score = s; ID = i; name = "Max";
 }
 Student (double s){
  score = s; ID = 555; name = "Aden";
 }
 void printInfo(){
  System.out.println(name + " (ID: "+ID+") scored "+ score );
 }
}
```

Important: Member methods can access all the data members directly, without the requirement of passing the data members through the formal parameter list.

```
public class MyClass{
public static void main(String[] xy){
 Student a = new Student(97.5);
 Student b = new Student();
 Student c = new Student("Isabel");
 Student d = new Student(333, 90);
 a.printInfo();b.printInfo();
 c.printInfo();d.printInfo();
 }
}
```

More on Constructors

- A constructor without parameters (e.g., Student ()) is called *no-argument constructor*, while consturctors with parameters are known as constructors with arguments.
- A class can be declared without constructors. In this case, a no-argument constructor with an empty body is declared in the class by the compiler. This constructor is known as *default constructor*. The default field values are as follows:
  - □ null for any reference type,
  - □ 0 for a numeric type,
  - □ false for a boolean type, and
  - □ '\u0000' for a char type. (it's the unicode for a blank space)
- However, Java assigns no default value to a local variable inside a method.

Example

Java assigns no default value to a local variable inside a method

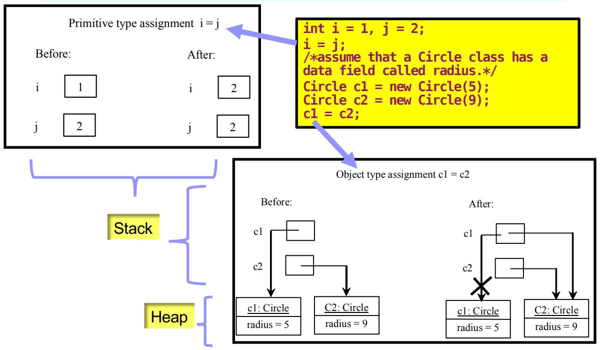
```
public class Test {
  public static void main(String[] args) {
    int x; // x has no default value
    String y; // y has no default value
    System.out.println("x is " + x);
    System.out.println("y is " + y);
  }
}
```

Compilation error:
variables/objects not initialized

Important info on objects

- An object (instance of a Class) can be used
  - □ in an array of objects,
  - □ in a method (both as a return-type and as a method parameter in the formal parameter list),
  - □ in a method call
- The objects can use assignment (=) operator.

Copying Variables of Primitive Data Types and Object Types



```
int i = 1, j = 2;
i = j;
/*assume that a Circle class has a
data field called radius.*/
Circle c1 = new Circle(5);
Circle c2 = new Circle(9);
c1 = c2;
```

Garbage Collection

- As shown on the previous slide:
  - □ After the assignment statement c1 = c2, c1 points to the same object referenced by c2.
  - □ The object previously referenced by c1 is no longer referenced. This object is known as garbage.
  - □ Garbage is automatically collected by JVM. (that is, like in C++, we are not recommended to release the memory)

Garbage Collection, cont

- *TIP: If you know that an object is no longer needed, you can explicitly assign null to a reference variable for the object. The JVM will automatically collect the space (release the memory) if the object is not referenced by any variable.*

Equality operator (==) OR .equals() method

- When comparing two object references (whether they are pointing to the same location or not), or two primitive data items, use equality (==) operator.
- When comparing two non-null object reference values, we can use .equals() method. .equals() method is automatically inherited from the Object class to any class, and available to use for any object. Discussion on Object Class
- Example:

Note: Always compare String objects with .equals method.

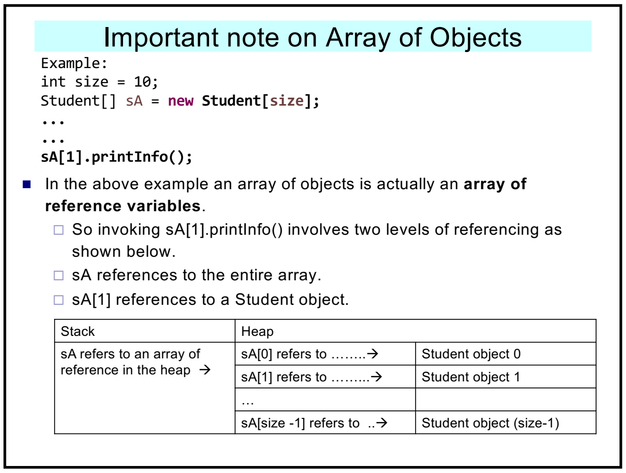
```
public class MyClass{
public static void main(String[] xy){
 String s1 = new String("abc");
 String s2 = new String("abc");
 if(s1 != s2)
   System.out.println("Unequal references");
 if(s1.equals(s2))
    System.out.println("Equal Values");
 }
}
```

Object Class

- http://docs.oracle.com/javase/10/docs/api/java/lang/Object.html
- In JAVA the class Object is the root class from where all the classes are inherited either implicitly or explicitly.
- One can use any method (equals(), toString(), wait(), hashCode() etc.) of the Object class using any class instance
- One can override any public non-final method () of the Object class in any class definition.

```
//Example: Array of objects
public class Main{
 public static void main(String[] args){
  Scanner input = new Scanner(System.in);
  System.out.print("Enter the size of the array: ");
  int size = input.nextInt();
  //Declaration and initialization
  Student[] sA = new Student[size];
  System.out.println("Let's initialize the array: ");
  for(int i = 0; i<sA.length; i++)
    sA[i] = new Student();//In-Class Discussion
  for(int i = 0; i<sA.length; i++) {//Populating the array
    System.out.print("Enter name "+(i+1)+": ");
    sA[i].name = input.next();
    System.out.print("Enter ID for " + sA[i].name +": ");
    sA[i].ID = input.nextInt();
    System.out.print("Enter score for " + sA[i].name +": ");
    sA[i].score = input.nextDouble();
  }
  System.out.println("Let's print the class list using printInfo() method: ");
  //Complete the code. This is your home work!
 }
}
```

```
public class Student {
 String name;
 int ID;
 double score;
 Student(){name = "Rebecca"; ID
= 111; score =100;}
void printInfo(){
  System.out.println(name + "
(ID: "+ID+") scored "+ score );
 }
}
```

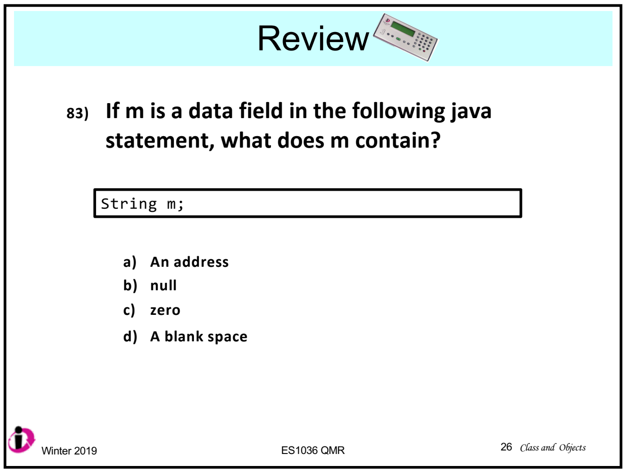


Important note on Array of Objects

```
Example:
int size = 10;
Student[] sA = new Student[size];
...
...
sA[1].printInfo();
```

- In the above example an array of objects is actually an **array of reference variables**.
  - So invoking sA[1].printInfo() involves two levels of referencing as shown below.
  - sA references to the entire array.
  - sA[1] references to a Student object.

Stack	Heap	
sA refers to an array of reference in the heap →	sA[0] refers to ……..→	Student object 0
	sA[1] refers to ………→	Student object 1
	…	
	sA[size -1] refers to ..→	Student object (size-1)

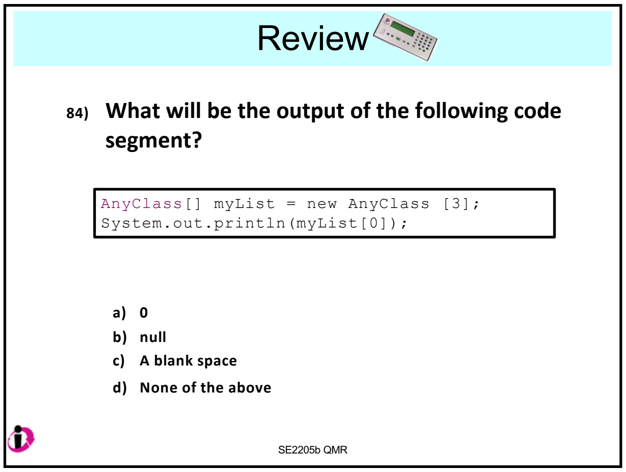


Review

83) If m is a data field in the following java statement, what does m contain?

```
String m;
```

- **a) An address**
- **b) null**
- **c) zero**
- **d) A blank space**

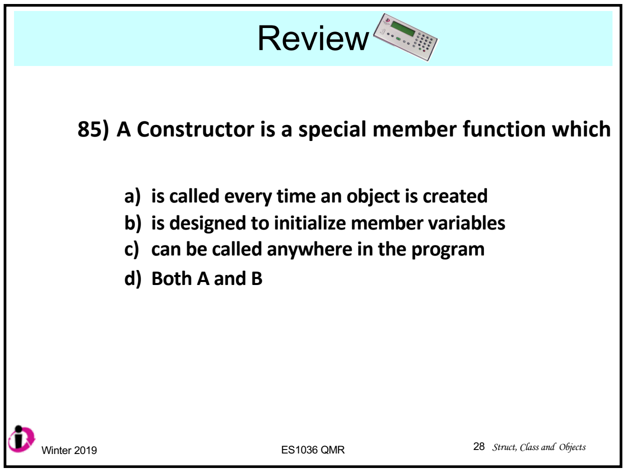


Review

84) What will be the output of the following code segment?

```
AnyClass[] myList = new AnyClass [3];
System.out.println(myList[0]);
```

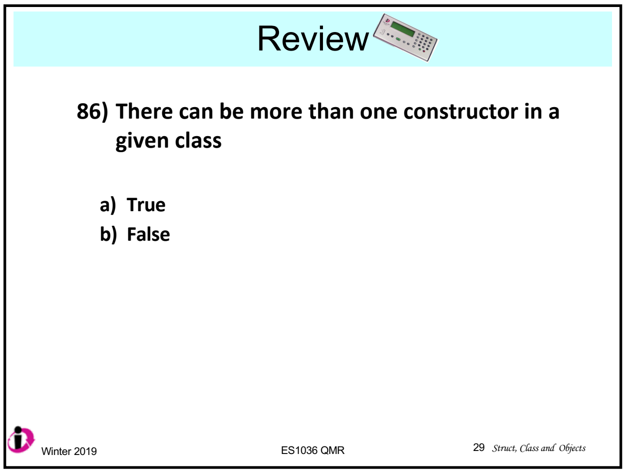
- **a) 0**
- **b) null**
- **c) A blank space**
- **d) None of the above**



Review

85) A Constructor is a special member function which

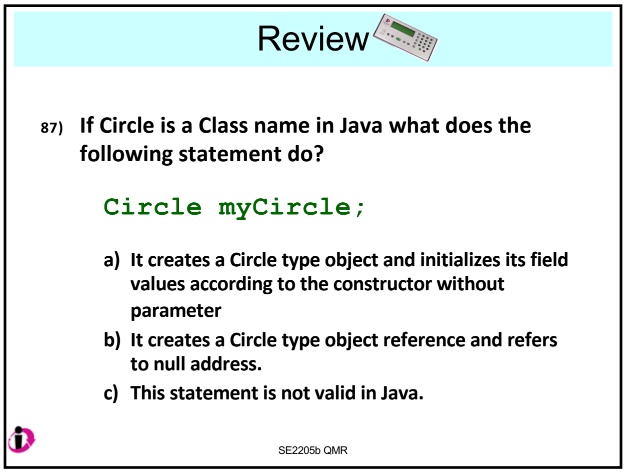
- **a) is called every time an object is created**
- **b) is designed to initialize member variables**
- **c) can be called anywhere in the program**
- **d) Both A and B**



Review

86) There can be more than one constructor in a given class

- **a) True**
- **b) False**



Review

87) If Circle is a Class name in Java what does the following statement do?

```
Circle myCircle;
```

- **a) It creates a Circle type object and initializes its field values according to the constructor without parameter**
- **b) It creates a Circle type object reference and refers to null address.**
- **c) This statement is not valid in Java.**

Access/Visibility Modifiers in a Class

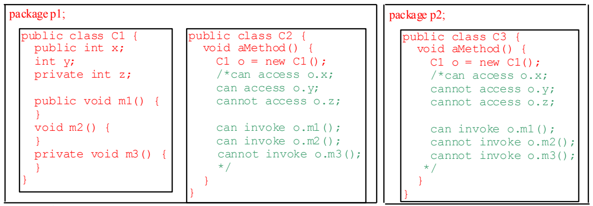
- The following keywords specify the accessibility of the class members
  - □ **public** class, data-fields and methods are visible to any class in any package.
  - □ **private** data-fields and methods can only be accessed by the members of the declaring class.
  - □ **protected** data-fields and methods can be accessed by the members of the declaring class, any class (discussed later) derived from the declaring class and any class within the same package.
- When no visibility modifier is used, **by default**, the class, data-fields, or methods can be accessed by any class in the same package.

Accessibility Summary

Modifier on members in a class	Accessed from the same class	Accessed from the same package	Accessed from a subclass	Accessed from a different package
public	✓	✓	✓	✓
protected	✓	✓	✓	-
default	✓	✓	-	-
private	✓	-	-	-

On Exam

- The private modifier restricts access to within a class, the default modifier restricts access to within a package, and the public modifier enables unrestricted access.



package p1;

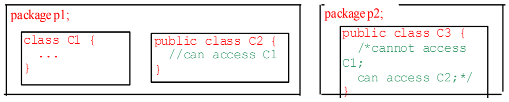
```
public class C1 {
  public int x;
  int y;
  private int z;

  public void m1() {
  }
  void m2() {
  }
  private void m3() {
  }
}
```

```
public class C2 {
  void aMethod() {
    C1 o = new C1();
    /*can access o.x;
    can access o.y;
    cannot access o.z;

    can invoke o.m1();
    can invoke o.m2();
    cannot invoke o.m3();
    */
  }
}
```

package p2;

```
public class C3 {
  void aMethod() {
    C1 o = new C1();
    /*can access o.x;
    cannot access o.y;
    cannot access o.z;

    can invoke o.m1();
    cannot invoke o.m2();
    cannot invoke o.m3();
   */
  }
}
```

- If a class does not have any access modifier (it is a default class), it can only be accessed with in the same package



package p1;

```
class C1 {
  ...
}
```

```
public class C2 {
  //can access C1
}
```

package p2;

```
public class C3 {
  /*cannot access
C1;
  can access C2;*/
}
```

Static methods with Objects in the signature and return-type

```
/*Problem: Write a static method that accepts a Student type object (as given
   below) and returns a Student type object*/
public class MyClass{
 public static void main(String[] args){//main() is called DRIVER method
  Student x = new Student();
  //calling a static method
  x = dataEntry(x);
  x.printInfo();
 }
 public static Student dataEntry(Student x) {
  x.ID = 111; x.name = "Albert"; x.score = 90;
  //One can input those values using scanner object
  return x;
 }
}
public class Student{
 String name; //Default visibility
 int ID; //Default visibility
 double score; //Default visibility
 Student(){name = "Amy"; ID = 111; score =100;}
 void printInfo(){//Default visibility
   System.out.println(name + " (ID: "+ID+") scored "+ score );
 }
}
```

private Access/Visibility Modifier in a Class

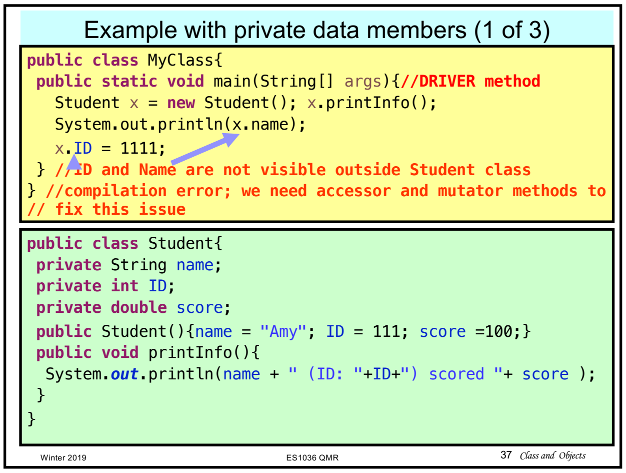
- Generally, in a class data fields (also know as data-members and instance-variables) are declared as private.
- Why?
  - □ To protect data.
  - □ To make code easy to maintain.
- To access or modify private data members, the class designer may add two types of public methods known as accessor and mutator methods.

Accessor and Mutator Methods

- An *accessor* method, also known as a *getter* or get-method, is a public method that gets the private field value available in the public domain. It has the following generic header:

  returnType get*DataMemberName*();
  - □ Here the returnType is the private data-field type that the programmer wants to reveal in the public domain.
- A mutator method, also known as as a *setter* or set-method, is a public method that offers the option to set the private field value from the public domain. This has the following generic header:

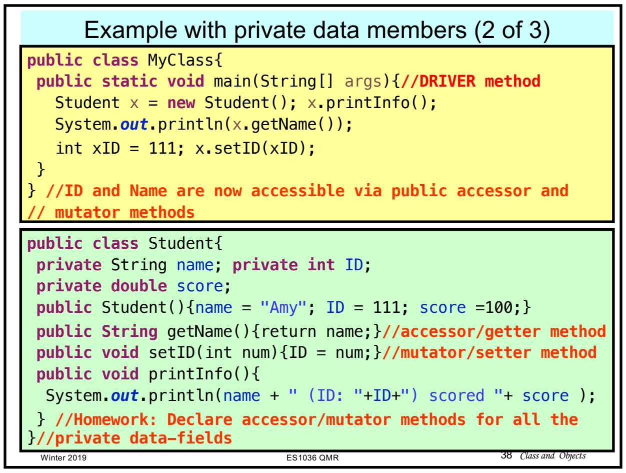
  void set*DataMemberName*(*dataType DataMemberValue*);
- *Note: The terms set and get with the method names are used for user friendliness. Any valid name can be used instead.*



Example with private data members (1 of 3)

```
public class MyClass{
 public static void main(String[] args){//DRIVER method
   Student x = new Student(); x.printInfo();
   System.out.println(x.name);
   x.ID = 1111;
 } //ID and Name are not visible outside Student class
} //compilation error; we need accessor and mutator methods to
// fix this issue
```

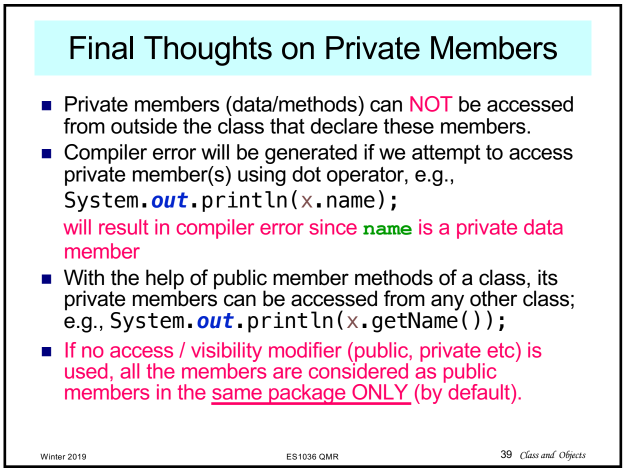
```
public class Student{
 private String name;
 private int ID;
 private double score;
 public Student(){name = "Amy"; ID = 111; score =100;}
 public void printInfo(){
  System.out.println(name + " (ID: "+ID+") scored "+ score );
 }
}
```



Example with private data members (2 of 3)

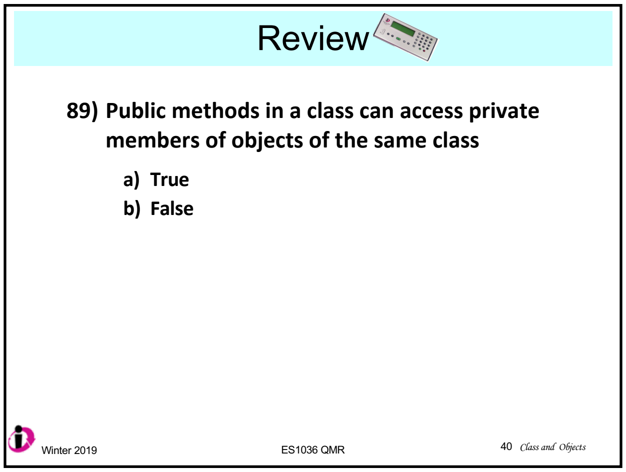
```
public class MyClass{
 public static void main(String[] args){//DRIVER method
   Student x = new Student(); x.printInfo();
   System.out.println(x.getName());
   int xID = 111; x.setID(xID);
 }
} //ID and Name are now accessible via public accessor and
// mutator methods
```

```
public class Student{
 private String name; private int ID;
 private double score;
 public Student(){name = "Amy"; ID = 111; score =100;}
 public String getName(){return name;}//accessor/getter method
 public void setID(int num){ID = num;}//mutator/setter method
 public void printInfo(){
  System.out.println(name + " (ID: "+ID+") scored "+ score );
 } //Homework: Declare accessor/mutator methods for all the
}//private data-fields
```



Final Thoughts on Private Members

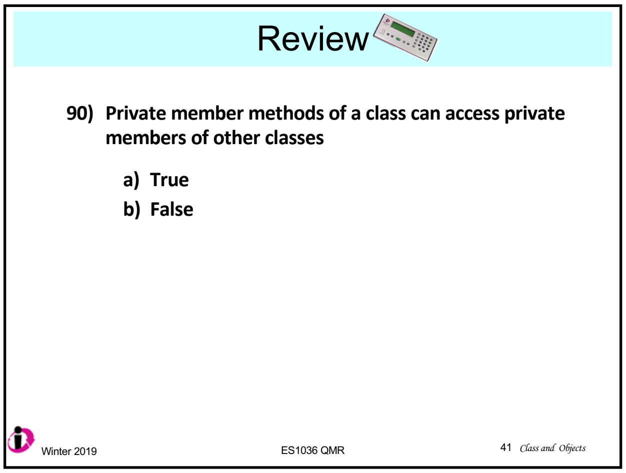
- Private members (data/methods) can NOT be accessed from outside the class that declare these members.
- Compiler error will be generated if we attempt to access private member(s) using dot operator, e.g., `System.out.println(x.name);` will result in compiler error since `name` is a private data member
- With the help of public member methods of a class, its private members can be accessed from any other class; e.g., `System.out.println(x.getName());`
- If no access / visibility modifier (public, private etc) is used, all the members are considered as public members in the same package ONLY (by default).



Review

89) **Public methods in a class can access private members of objects of the same class**

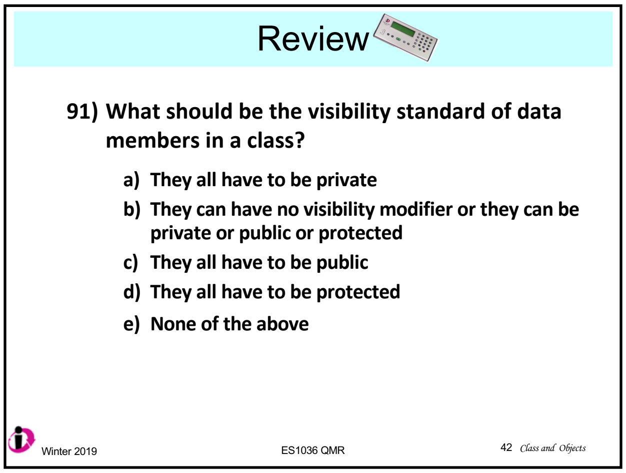
a) **True**
b) **False**



Review

90) **Private member methods of a class can access private members of other classes**

a) **True**
b) **False**



Review

91) **What should be the visibility standard of data members in a class?**

a) **They all have to be private**
b) **They can have no visibility modifier or they can be private or public or protected**
c) **They all have to be public**
d) **They all have to be protected**
e) **None of the above**

Review

11) If a class in java does not contain any visibility modifier (such as private, public, protected) can that class be used by any other class from the other packages?

a) Yes
b) No

Review

93) When passing a parameter by value, the parameter is copied to a new object

a) True
b) False

Review

94) When passing a parameter ‘by value’ to a method, changes to this parameter inside the method affects the original object.

a) True
b) False

Review

95) When passing an object by reference-value during a method-call, changes to this object inside the called method affects the original object in the calling method.

a) True
b) False

Review

96) Output?

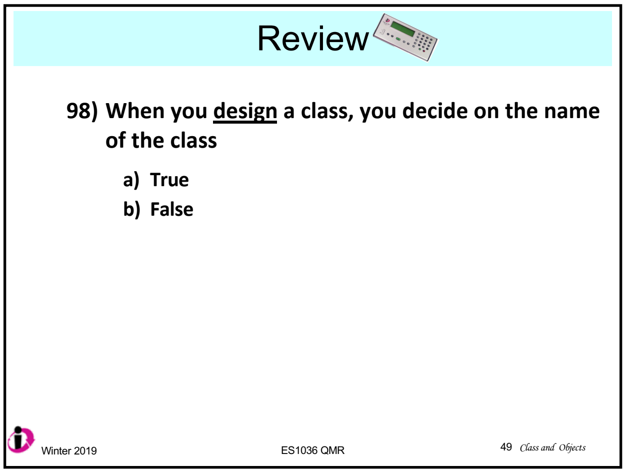
```
 Public class A{
    int x; int y; int z;
    A(){
         x = 1; y = 2; z = 3;
    }
  }
public class MyClass{
  public static void main(String[] args){
    A a();
    System.out.println(a.x+" "+a.y+" "+a.z);
  }
}
```

a) 2 2 2
b) 1 1 1
c) 3 3 3
d) 1 2 3

Review

97) The ________ operator can be used to assign one object to another.

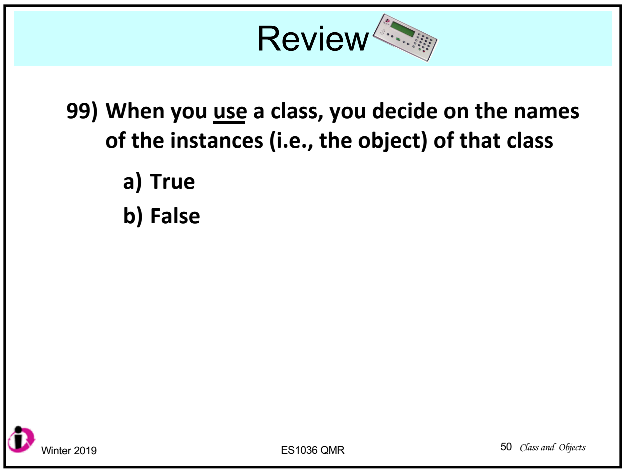
a) equality or equivalent-to (==)
b) assignment (=)
c) address (&)
d) None of the above



Review

98) When you <u>design</u> a class, you decide on the name of the class

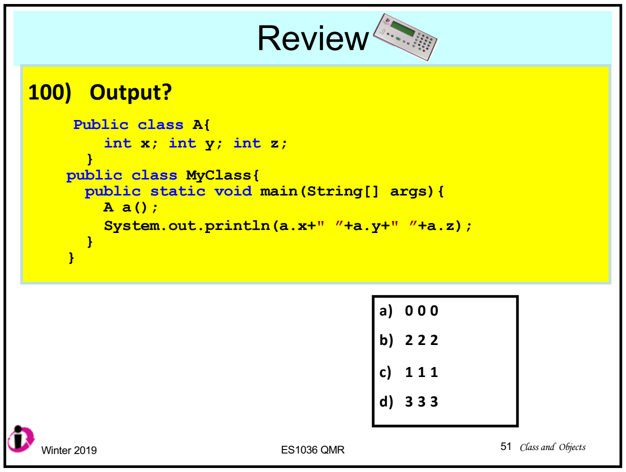
a) True
b) False



Review

99) When you <u>use</u> a class, you decide on the names of the instances (i.e., the object) of that class

a) True
b) False

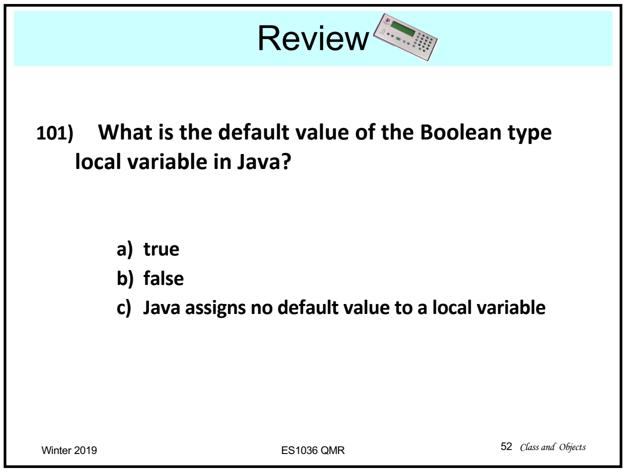


Review

100) Output?

```
 Public class A{
    int x; int y; int z;
   }
public class MyClass{
  public static void main(String[] args){
    A a();
    System.out.println(a.x+" ″+a.y+" ″+a.z);
   }
}
```

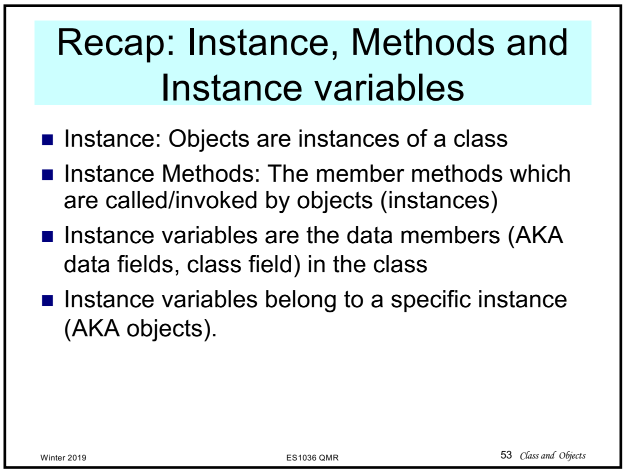
a) 0 0 0
b) 2 2 2
c) 1 1 1
d) 3 3 3



Review

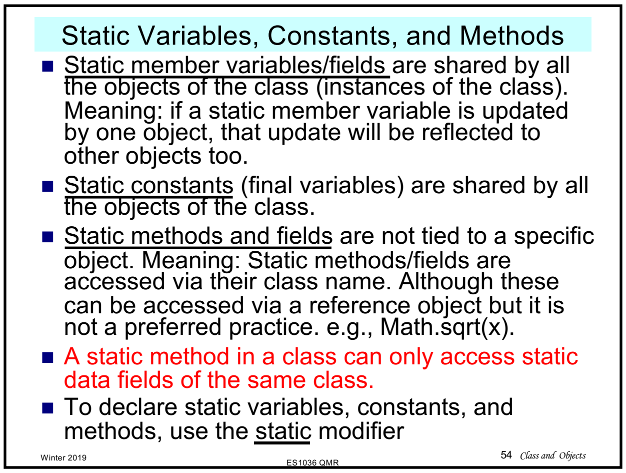
101) What is the default value of the Boolean type local variable in Java?

a) true
b) false
c) Java assigns no default value to a local variable



Recap: Instance, Methods and Instance variables

- Instance: Objects are instances of a class
- Instance Methods: The member methods which are called/invoked by objects (instances)
- Instance variables are the data members (AKA data fields, class field) in the class
- Instance variables belong to a specific instance (AKA objects).



Static Variables, Constants, and Methods

- <u>Static member variables/fields</u> are shared by all the objects of the class (instances of the class). Meaning: if a static member variable is updated by one object, that update will be reflected to other objects too.
- <u>Static constants</u> (final variables) are shared by all the objects of the class.
- <u>Static methods and fields</u> are not tied to a specific object. Meaning: Static methods/fields are accessed via their class name. Although these can be accessed via a reference object but it is not a preferred practice. e.g., Math.sqrt(x).
- A static method in a class can only access static data fields of the same class.
- To declare static variables, constants, and methods, use the <u>static</u> modifier

Example code: Static variable and method (1 of 2)

```
public class MyClass{
 public static void main(String[] args){//main() is called DRIVER method
  Student x = new Student();
  System.out.println("Number of student objects: "+Student.getNumberOfObject());
  Student y = new Student(78.9);
  System.out.println("Number of student objects: "+y.getNumberOfObject());
  System.out.println("Number of student objects: "+x.numberOfObjects);
  Student[] sa = new Student[4];
  for(int i = 0; i<sa.length; i++)
    sa[i] = new Student();
  System.out.println("Number of student objects: "+Student.numberOfObjects);//6
  System.out.println("Number of student objects: "+sa[0].numberOfObjects);//6
  System.out.println("Number of student objects: "+y.numberOfObjects);//6
 }
}
```

```
public class Student{
 private double score;
 public static int numberOfObjects = 0;
 public Student(){score =100; numberOfObjects++;}
 public Student(double sc){score = sc; numberOfObjects++;}
 public double getScore() {return score;}
 public void setScore(double sc){score = sc;}
 public static int getNumberOfObject() {
   return numberOfObjects;
 }
}//If there any accessor or mutator method in the Student class?
```

Take-aways from the Example code (2 of 2)

- There is one static field (`public static int numberOfObjects`) and one static method (`public static int getNumberOfObject()`) in the Student class.
- The static field numberOfObjects is shared by all the objects, and it keeps track of the number of Student objects created.
- Static method has been called via the class name (`Student.getNumberOfObject()`) which is preferred over calling via an object name (`y.getNumberOfObject()`). Both calls are valid calls.

Another example with Static data field

```
public class MyClass{
 static int counter = 0;
 public static void main(String[] args){
 doSomething(3);
 //same as: MyClass.doSomething(3)
 doSomething(3);
 doSomething(3);
 System.out.println(counter);
}
public static void doSomething(int x) {
 counter++;
 x++;
 System.out.println(counter+", "+x);
 }
}/*What will happen if we remove the static modifier from the variable counter? */ See slide #53.
```

In Myclass:

- How many fields/data members do u see?
- How many methods do you see?
- Output?

Another example with static methods/data fields

```
package testpackage;
public class anyClass {
public static void main(String[] args){
    Circle c1 = new Circle();
    System.out.println("c1 radius: " +
   c1.radius +
    " and number of Circle objects " +
   Circle.numberOfObjects);
    Circle c2 = new Circle(5);
    c1.radius = 9;
    System.out.println("c1 radius: " +
   c1.radius +
    " and number of Circle objects " +
   c1.numberOfObjects);
    System.out.println("c2 radius: " +
   c2.radius +
    " and number of Circle objects " +
   Circle.numberOfObjects);
    }
}
```

```
class Circle {
  double radius;
  static int numberOfObjects = 0;
  Circle() {
    radius = 1.0;
    numberOfObjects++;
  }
  Circle(double newRadius) {
    radius = newRadius;
    numberOfObjects++;
  }
  static int getNumberOfObjects() {
    return numberOfObjects;
  }
  double getArea() {
    return radius * radius * Math.PI;
  }
}
```

In Circle Class:

- How many fields/data members do u see?
- How many methods in total do you see? How many constructors?
- What is the visibility characteristic of the data filed radius?
- Output?

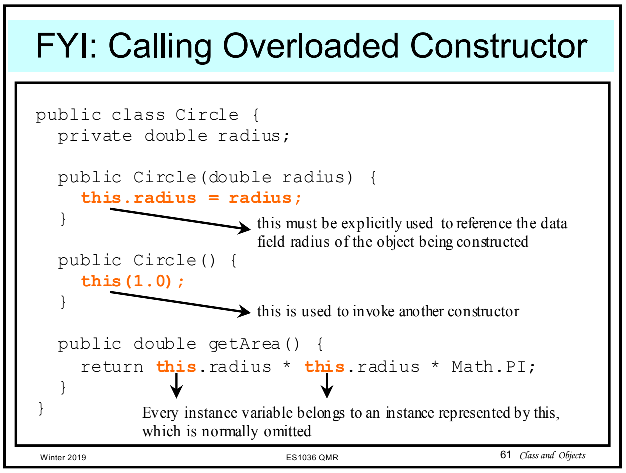
FYI: The 'this' Keyword

- The this keyword is the name of a reference that refers to an object itself.
- One common use of the this keyword is to refer to a class's *data fields*.
- Another common use of the this keyword is to enable a constructor to invoke another constructor of the same class.

FYI: Reference the Hidden Data Fields

```
public class Foo {
  int i = 5;
  static double k = 0;

  void setI(int i) {
    this.i = i;
  }

  static void setK(double k) {
    Foo.k = k;
  }
}
```

```
Suppose that f1 and f2 are two objects of Foo.

   f1.setI(10); /* it executes
   this.i = 10, where this refers f1*/

   f2.setI(45); /* it executes
   this.i = 45, where this refers f2*/
```



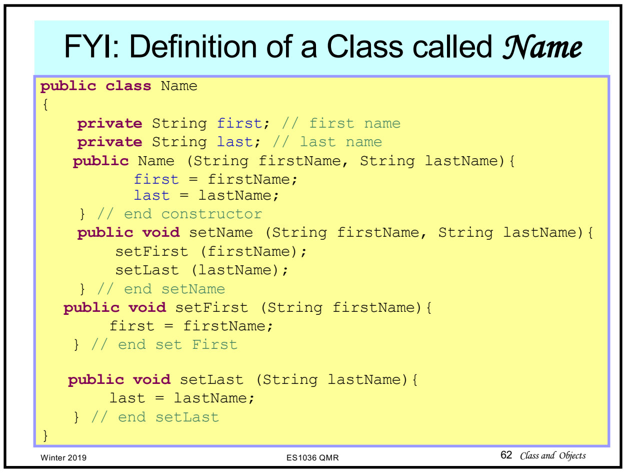
FYI: Calling Overloaded Constructor

```
public class Circle {
  private double radius;

  public Circle(double radius) {
    this.radius = radius;
  }

  public Circle() {
    this(1.0);
  }

  public double getArea() {
    return this.radius * this.radius * Math.PI;
  }
}
```

this must be explicitly used to reference the data field radius of the object being constructed

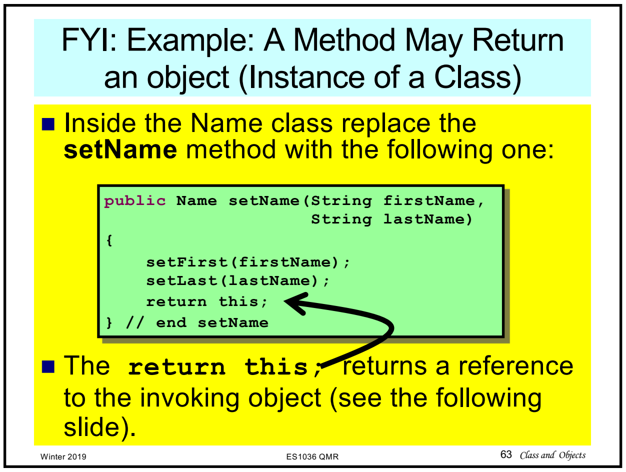
this is used to invoke another constructor

Every instance variable belongs to an instance represented by this, which is normally omitted



FYI: Definition of a Class called *Name*

```
public class Name
{
    private String first; // first name
    private String last; // last name
   public Name (String firstName, String lastName){
         first = firstName;
         last = lastName;
    } // end constructor
    public void setName (String firstName, String lastName){
        setFirst (firstName);
        setLast (lastName);
    } // end setName
  public void setFirst (String firstName){
       first = firstName;
   } // end set First

   public void setLast (String lastName){
       last = lastName;
   } // end setLast
}
```

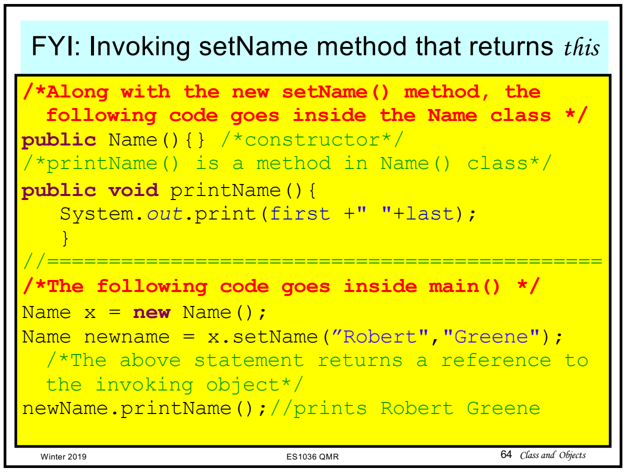


FYI: Example: A Method May Return an object (Instance of a Class)

- Inside the Name class replace the **setName** method with the following one:

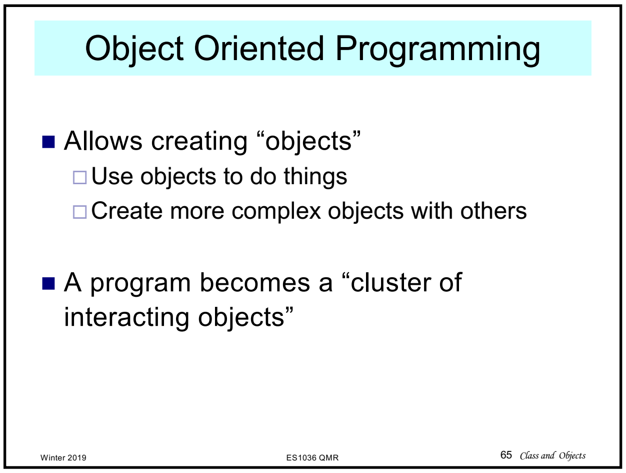
```
public Name setName(String firstName,
                    String lastName)
{
    setFirst(firstName);
    setLast(lastName);
    return this;
} // end setName
```

- The **`return this;`** returns a reference to the invoking object (see the following slide).



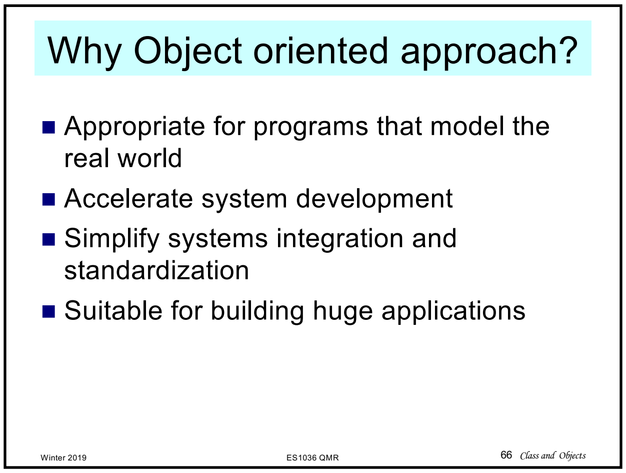
FYI: Invoking setName method that returns *this*

```
/*Along with the new setName() method, the
  following code goes inside the Name class */
public Name(){} /*constructor*/
/*printName() is a method in Name() class*/
public void printName(){
   System.out.print(first +" "+last);
   }
//============================================
/*The following code goes inside main() */
Name x = new Name();
Name newname = x.setName(”Robert","Greene");
  /*The above statement returns a reference to
  the invoking object*/
newName.printName();//prints Robert Greene
```



Object Oriented Programming

- Allows creating “objects”
  - □ Use objects to do things
  - □ Create more complex objects with others
- A program becomes a “cluster of interacting objects”



Why Object oriented approach?

- Appropriate for programs that model the real world
- Accelerate system development
- Simplify systems integration and standardization
- Suitable for building huge applications