

Explicit constructor invocation

```
public class CourseManagement {  
    public static void main() {  
        Student s1=new Student();  
        Student s2=new Student("Jagan", "13CS04");  
        Student s3=new  
        Student("Jyoti","13CS05","CS201", "CS251",  
        "CS261");  
    }  
}
```

Caution: There is a difference between a constructor and method.

- Constructors can only be called in conjunction with a new operator
- We cannot apply a constructor on an existing object to reset its fields, i.e.
s2.Student("Jagan", "11ME05") is not allowed

Testing Objects for Equality

- Using the == operator (continued):
 - The == operator returns true if the two reference variables point to the same object; i.e., the two reference variables contain the same address. For example, what does this code fragment print?

```
Car car1 = new Car();
Car car2 = car1;
if (car1 == car2)
{
    System.out.println("the same");
}
else
{
    System.out.println("different");
}
```

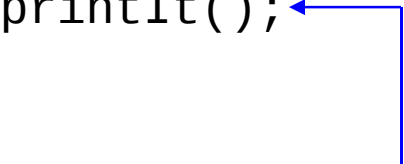
Passing References as Arguments

- Suppose you pass a reference variable to a method, and inside the method you update the reference variable's instance variables. What happens? ...
- Remember that a reference variable holds the address of an object, not the object itself.
- So in passing a reference variable argument to a method, a copy of the object's address (not a copy of the object itself) is passed to the method and stored in the method's parameter.
- Since the parameter and the argument hold the same address value, they point to the same object. Thus, if one of the parameter's instance variables is updated, then the update will simultaneously update the argument's instance variable in the calling module.

Method-Call Chaining

```
public class Car3Driver
{
    public static void main(String[] args)
    {
        Car3 car = new Car3();

        car.setMake("Toyota").setYear(2008).printIt();
    } // end main
} // end class Car3Driver
```



a method-call chain

Method-Call Chaining

```
public class Car3
{
    private String make;
    private int year;

    /*******

public Car3 setMake(String make)
{
    this.make = make;
    return this;
} // end setMake

public Car3 setYear(int year)
{
    this.year = year;
    return this;
} // end setYear

/*******

public void printIt()
{
    System.out.println(make + ", " + year);
} // end printIt
} // end class Car3
```

The return type is the same as the class name.

Return the calling object.

Class Methods

- If you have a method that accesses class variables and not instance variables
 - Declare the method to be a *class method*
 - Add `static` to the method's heading like this:
`<private-or-public> static <return-type> <method-name>(<parameters>)`

- **Example:**

```
public class Mouse
{
    private static int mouseCount;
    private static double averageLifeSpan;

    public static void printMouseCount()
    {
        System.out.println("Total mice = " +
            Mouse.mouseCount);
    }
}
```

To access a class variable, prefix it with `<class-name>` dot.

Adding Elements to an ArrayList Object

- To add an element to the end of an ArrayList object, use this syntax:

ArrayList-reference-variable.add(item);

- The *item* that's added must be the same type as the type specified in the ArrayList's declaration.

- Write a code fragment that creates this ArrayList object:

computerScientists	
0	"Ada Lovelace"
1	"Grace Hopper"
2	"Carol Bartz"

How to Update an ArrayList Object

- Draw a picture of the colors ArrayList after this code fragment executes:

```
String mixedColor;  
ArrayList<String> colors = new ArrayList<String>();  
colors.add("red");  
colors.add("green");  
colors.add("blue");  
mixedColor = colors.get(0) + colors.get(1);  
colors.set(2, mixedColor);
```

Storing Primitives in an ArrayList

- As mentioned previously, ArrayLists store references. For example, in the Survivor program, tribe is an ArrayList of strings, and strings are reference types.
- If you need to store primitives in an ArrayList, you can't do it directly, but you can do it if the primitives are wrapped up in wrapper classes.
- Ever since Java 5.0, the "wrapping up" process has been done behind the scenes. For ArrayLists, it's done automatically if a wrapper class is used in an ArrayList declaration.
- The StockAverage program on the next slide reads `int` stock values and stores them in an ArrayList. After all stock values are entered, the program calculates the average stock value.
- Why is an ArrayList appropriate for calculating a stock average?