Object-Oriented Programming

Chapter Two : The inside of objects and classes: More on OOP concepts

Contents

- member methods and their components
- instantiation and initializing class objects
- constructors
  - default and parameterized
  - overloaded constructors
- methods
- access specifiers
- accessors and mutators
- calling and returning methods
- static and instance members

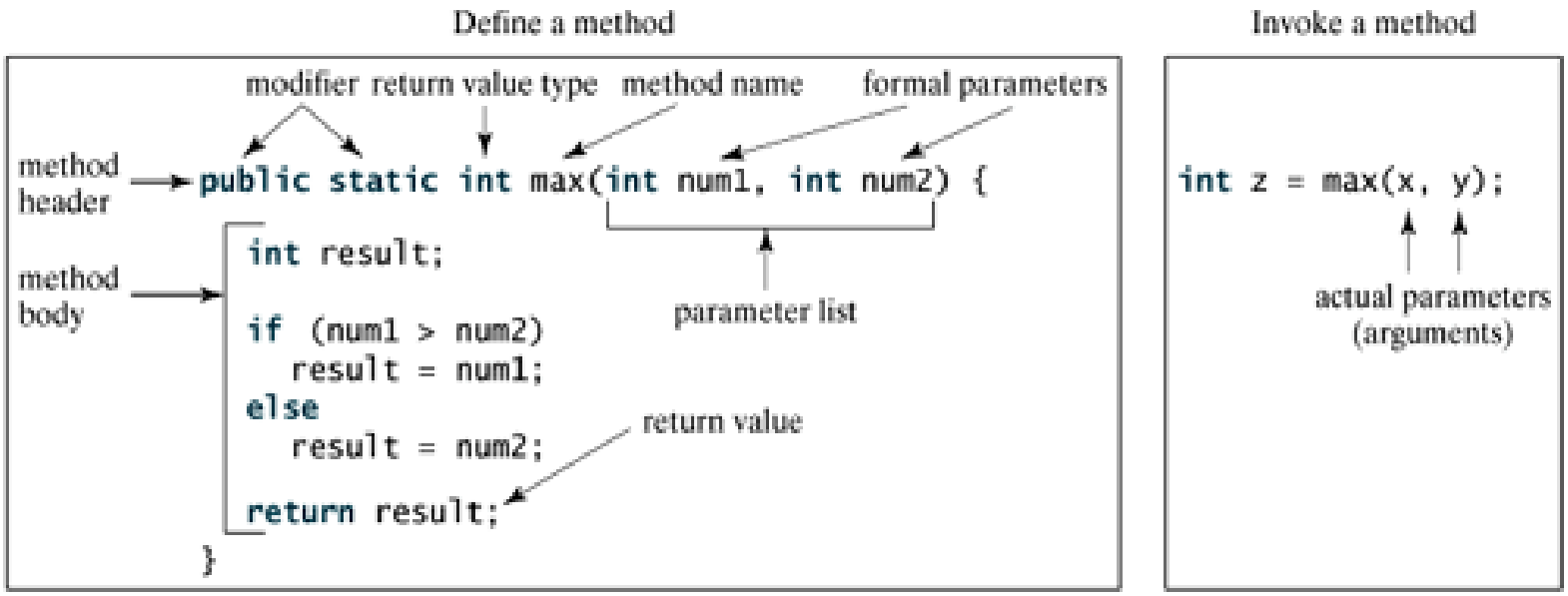
member methods and their components

- What is method ?
  - A method is a collection of statements that are grouped together to perform an operation.
  - Example:
    - System.out.println()
    - OptionPane.showMessageDialog()
    - JOptionPane.showInputDialog()
    - Integer.parseInt(), Double.parseDouble()
    - Math.pow()
    - Math.random()
-

member methods and their components Cont.

Creating a Method

- a method created to find which of two integers is bigger.
- This method, named max, has two int parameters, num1 and num2, the larger of which is returned by the method. Figure 2.1 illustrates the components of this method.



member methods and their components Cont.

- The method header specifies the *modifiers*, *return value type*, *method name*, and *parameters* of the method. The modifier, which is optional, tells the compiler how to call the method.
- A method may return a value. The returnValueType is the data type of the value the method returns.
- Some methods perform the desired operations without returning a value. In this case, the returnValueType is the keyword void.
- The variables defined in the method header are known as *formal parameters or simply parameters*.
- A parameter is like a placeholder. When a method is invoked, you pass a value to the parameter. This value is referred to as *actual parameter* or *argument*.

member methods and their components Cont.

- The parameter list refers to the type, order, and number of the parameters of a method.
- The method name and the parameter list together constitute the *method signature*.
- Parameters are optional; that is, a method may contain no parameters.
- The method body contains a collection of statements that define what the method does.
- The method body of the max method uses an if statement to determine which number is larger and return the value of that number.
- A return statement using the keyword return is required for a non void method to return a result.
- The method terminates when a return statement is executed.

Instantiation and initializing class objects

- Instance variables and methods (also called members) are those associated with individual instances (or objects) of a class
- Instance variables belong to a specific instance.
- Instance methods are invoked by an instance of the class.
- To get to the value of an instance variable or to invoke instance methods, you use an expression in what's called dot notation.
- Example
- Circle cl = new Circle ();//instantiation class Circle object
- cl.radius = 5;
- System.out.println("Area " + cl.getArea());

Constructors

- Constructors are used to initialize the instance variables (fields) of an object.
- Constructors are **similar** to methods, but with important differences.
- Constructor name is class name

  - A constructors must have the same name as the class its in.

Default constructor

- If you don't define a constructor for a class, a default parameterless constructor is automatically created by the compiler.
- The default constructor calls the default parent constructor (super()) and initializes all instance variables to default value (zero for numeric types, null for object references, and false for Booleans).
- **Constructors** are invoked using the new operator when an object is created. Constructors play the role of initializing objects.

Constructors Cont.

- **Differences between methods and constructors**.
  - There is *no return type* given in a constructor signature (header).
    - ✓ The value is this object itself so there is no need to indicate a return value.
  - There is *no return statement* in the body of the constructor.
  - The *first line* of a constructor must either be a call on *another constructor in the same class (using this)*, or a call on the super class constructor (using super).
    - ✓ If the first line is neither of these, the compiler automatically inserts a call to the parameterless super class constructor.

Constructors Cont.

```
1  package animal;
2  public class Animal {
3      String animalName;
4      public Animal()
5      {
6
7      }
8      public void setAnimalName(String name )
9      {
10         animalName=name;
11     }
12     public void getAnimal()
13     {
14         System.out.println(animalName);
15     }
16     public static void main(String[] args) {
17         Animal an=new Animal();
18         an.setAnimalName("Lion");
19         an.getAnimal();
20     }
21 }
```

Constructor with parameterless

Method with one parameter

Method with parameterless

```
Lion
BUILD SUCCESSFUL (total time: 0 seconds)
```

Constructors Cont.

- Objects are created from classes using new operator
  - ClassName cn = new ClassName([parameters]); or
  - -ClassName cn;

    cn = new ClassName ([parameters]);
- The new operator creates an object of type ClassName
- The object may be created using any of its constructors
  - if it has constructors other than the default one appropriate parameters must be provided to create objects using them.

Constructors Cont.

```
1   package animal;
2   public class Animal {
3       String animalName1;
4        String animalName2;
5        public Animal(String name)
6        {
7            animalName1=name;
8        }
9        public void setAnimalName(String name )
10       {     animalName2=name;
11       }
12       public void getAnimal()
13       {    System.out.println(animalName1);
14           System.out.println(animalName2);
15       }
16       public static void main(String[] args) {
17           Animal an=new Animal("Dog");
18           an.setAnimalName("Lion");
19           an.getAnimal();
20       }
21  }
```

Constructor with One parameter

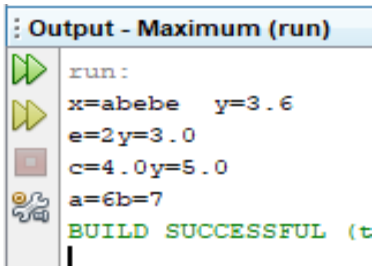
Method with One parameter

Method with no parameter

```
run:
Dog
Lion
BUILD SUCCESSFUL (total time: 0 seconds)
```

Overloaded constructors

```
1  package maximum;
2  public class DisplayResult {
3      public DisplayResult(int a, int b)
4      {
5          System.out.println("a="+a+"b="+b);
6      }
7      public DisplayResult(double c, double d)
8      {
9          System.out.println("c="+c+"y="+d);
10     }
11     public DisplayResult(int e, double f)
12     {
13         System.out.println("e="+e+"y="+f);
14     }
15     public DisplayResult(String x, double y)
16     {
17         System.out.println("x="+x+"   y="+y);
18     }
19     public static void main(String[] args) {
20          DisplayResult max4=new  DisplayResult("abebe",3.6);
21         DisplayResult max1=new  DisplayResult(2,3.0);
22         DisplayResult max2=new  DisplayResult(4.0,5.0);
23         DisplayResult max3=new  DisplayResult(6,7);
24     }
25 }
```



Overloaded constructors

- The DisplayResult constructor that was used earlier works on multiple times in the above program but with different data type parameter list .
- To create another method with the same name but different parameters, this is called **overloaded** :
- If you DisplayResult constructor create an object with int parameters, the DisplayResult constructor that expects int parameters will be invoked;
- if you create DisplayResult constructor with double parameters, the DisplayResult constructor that expects double parameters will be invoked.
- This is referred to as method overloading; that is, two methods have the same name but different parameter lists within one class.
- The Java compiler determines which method is used based on the method signature.

Class Methods

- Class methods, like class variables, apply to the class as a whole and not to its instances.
- Class methods are commonly used for general utility methods that may not operate directly on an instance of that class, but fit with that class **conceptually**.
- For example, the String class contains a class method called valueOf(), which can take one of many different types of arguments.

➢ The valueOf() method then returns a new instance of String containing the string value of the argument it was given. This method doesn't operate directly on an existing instance of String, but getting a string from another object or data type is definitely a String-like operation, and it makes sense to define it in the String class.

Class Methods Cont.

- For example, the class method Math.max() takes two arguments and returns the larger of the two. You don't need to create a new instance of Math; just call the method anywhere you need it, like this:

```
int biggerOne = Math.max(x, y);
```

- To call a class method, you use dot notation as you do with instance methods.

```
String s, s2;
s = "foo";
s2 = s.valueOf(5);
 s2 = String.valueOf(5);
```

Modifiers in Java

- Modifiers allow programmers to control the behavior of classes, object constructors, methods, variables, and even blocks of code.
- Combinations of modifiers may be used in meaningful ways.
- Modifiers are applied by prefixing the appropriate keyword for the modifier to the declaration of the class, variable or method.
- The utility of some of these modifiers may not be immediately apparent.
- However, later in the lectures, we will see situations where they are necessary.

Modifiers in Java Cont.

Modifier	Used on	Meaning
Abstract	class	Contains unimplemented methods and cannot be instantiated.
	interface	All interfaces are abstract. Optional in declarations
	method	No body, only signature. The enclosing class is abstract
Final	class	Cannot be subclassed
	method	Cannot be overridden and dynamically looked up
	field	Cannot change its value. static final fields are compile-time constants.
	variable	Cannot change its value.
native	method	Platform-dependent. No body, only signature

Modifiers in Java Cont.

None (package)	class	Accessible only in its package
	interface	Accessible only in its package
	member	Accessible only in its package
static	class	Make an inner class top-level class
	method	A class method, invoked through the class name.
	field	A class field, invoked through the class name one instance, regardless of class instances created.
	initializer	Run when the class is loaded, rather than when an instance is created.
private	member	Accessible only in its class(which defines it).

Modifiers in Java Cont.

public	class	Accessible anywhere
	interface	Accessible anywhere
	member	Accessible anywhere its class is.
protected	member	Accessible only within its package and its subclasses

synchronized	method	For a static method, a lock for the class is acquired before executing the method. For a non-static method, a lock for the specific object instance is acquired.
transient	field	Not be serialized with the object, used with object serializations.
volatile	field	Accessible by unsynchronized threads, very rarely used.

Modifiers in Java Cont.

- The table(matrix) in the next slide shows java modifiers indicating summary of which modifier to which element can be applied.

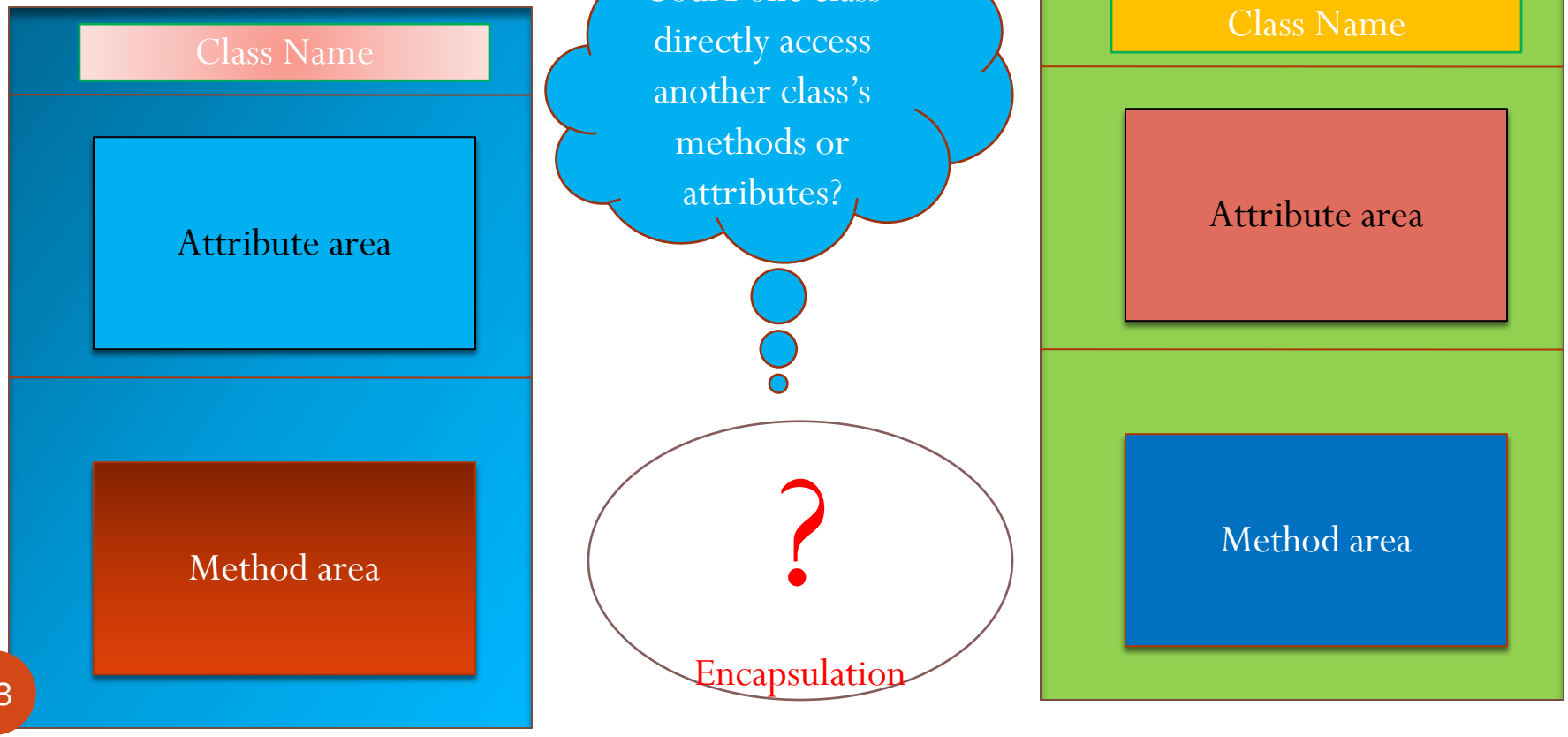
Modifiers in Java Cont.

Modifiers-Elements Matrix in Java

element / modifier	Data field	Method	Constructor	Class		Interface	
				top level (outer)	nested (inner)	top level (outer)	nested (inner)
abstract	no	yes	no	yes	yes	yes	yes
final	yes	yes	no	yes	yes	no	no
native	no	yes	no	no	no	no	no
private	yes	yes	yes	no	yes	no	yes
protected	yes	yes	yes	no	yes	no	yes
public	yes	yes	yes	yes	yes	yes	yes
static	yes	yes	no	no	yes	no	yes
synchronized	no	yes	no	no	no	no	no
transient	yes	no	no	no	no	no	no
volatile	yes	no	no	no	no	no	no

Access modifiers

- Access modifiers determine the visibility or accessibility of an object's attributes and methods to other objects. Public private, protected



Access modifiers Cont.

- Class's public members are accessible wherever the program has a reference to an object of that class or one of its subclasses.
- A class's private members are accessible only from within the class itself.
- A class's protected member can be accessed by members of that superclass, by members of its subclasses and by members of other classes in the same package (i.e., protected members also have package access).
- In Java related classes are grouped into package.
  - Java contains many predefined classes that are grouped into categories of related classes called packages. Together, we refer to these packages as the Java Application Programming Interface (Java API), or the Java class library.

Modifiers Rules

- A method declaration can contain only one of the access modifiers public, protected and private
- Classes cannot be declared abstract and final simultaneously
- Abstract methods cannot be declared private, static, final, native, or synchronized
- Abstract and native methods have no body
  - abstract void method();
  - native void method();
- A class that contains abstract method(s) must be declared abstract
- Final fields cannot be volatile

This keyword

- Use this to refer to the object that invokes the instance method.
- Use this to refer to an instance data field.
- Use this to invoke an overloaded constructor of the same class.

```
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

Invoking f1.setI(10) is to execute
  f1.i = 10, where this is replaced by f1

Invoking f2.setI(45) is to execute
  f2.i = 45, where this is replaced by f2
```

This keyword…

```
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

This keyword…

- Note that:
  - Class methods ***cannot*** access instance variables or instance methods directly—they must use an object reference.
  - Class methods cannot use the this keyword as there is no instance for this to refer to.

Get and Set methods

- A class's private fields can be manipulated only by methods of that class.
- So a client of an object—that is, any class that calls the object's methods—calls the class's public methods to manipulate the private fields of an object of the class.
- Classes often provide public methods to allow clients of the class to set (i.e., assign values to) or get (i.e., obtain the values of) private instance variables.
- The names of these methods need not begin with *set or get, but this naming convention* is highly recommended in Java

Get and Set methods Cont.

- Colloquially, a get method is referred to as a getter (or accessor), and a set method is referred to as a setter (or mutator).
- In general, the accessor method is named get<PropertyName>(), which takes no parameters and returns a primitive type value or an object of a type identical to the property type. For example,

```
public String getMessage() { }
```

- For a property of boolean type, the accessor method should be named is<PropertyName>(), which returns a boolean value. For example,

```
public boolean isCentered() { }
```

Get and Set methods

- The mutator method should be named set<PropertyName>(dataType p), which takes a single parameter identical to the property type and returns void. For example,

```
public void setMessage(String s) { }
 public void setCentered(boolean centered) { }
```

Example

```
import java.text.SimpleDateFormat;
import java.text.ParseException;
import java.util.Date;
public class Student {
    private String firstName;
    private String lastName;
    private String sex;
    private String department;
    private Date birthDate;

    public Student() {
        firstName="";
        lastName="";
        sex="";
        department="";
        birthDate=new Date();
    }
   public Student(String firstName,String lastName,String sex,String department,String birthDate) {
        this.firstName=firstName;
        this.lastName=lastName;
        this.sex=sex;
        this.department=department;
        this.birthDate=new Date(birthDate);
    }
```

Class Name

Data Filelds

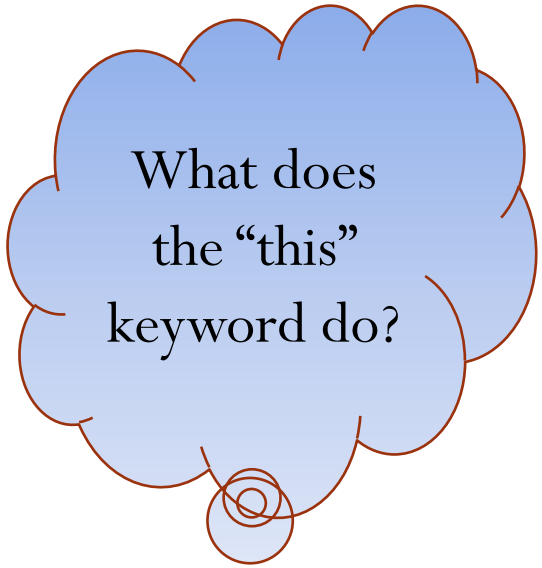
Constructors

Initializing attributes

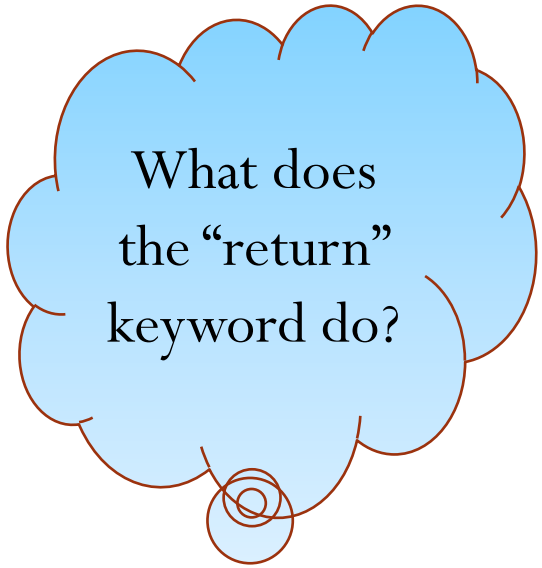
```
public void setFirstName(String firstName)
{
    this.firstName=firstName;
}
public void setLastName(String lastName)
{
    this.lastName=lastName;
}
public void setSex(String sex)
{
    this.sex=sex;
}
public void setDepartment(String department)
{
    this.department=department;
}
public void setBirthDate(String birthDate)throws ParseException
{

    SimpleDateFormat df=new SimpleDateFormat("dd/MM/yyyy");
    this.birthDate = df.parse(birthDate);

}
```



```
public String getFirstName()
{
    return this.firstName;
}
public String getLastName()
{
    return this.lastName;
}
public String getSex()
{
    return this.sex;
}
public String getDepartment()
{
    return this.department;
}
public Date getBirthDate()
{
    return this.birthDate;
}
```



```java
package StudentInformation;
import java.text.ParseException;
public class StudentInformation {
    public static void main(String[] args) throws ParseException {
        Student st1=new Student("Aster","Marew","F","Software Engineering ","02/08/2008");
        Student st2=new Student();
        st2.setFirstName("Ali");
        st2.setLastName("Mohammed");
        st2.setSex("M");
        st2.setdepartment("Computer Science");
        st2.setBirthDate("26/12/2006");
        System.out.println("Student's Name"+st1.getFirstName());
        System.out.println("father's Name:"+st1.getLastName());
        System.out.println("sex "+st1.getSex());
        System.out.println("Department Name:"+st1.getDepartment());
        System.out.println("Birth Date:"+st1.getBirtDate());
        System.out.println("Student's Name"+st2.getFirstName());
        System.out.println("father's Name:"+st2.getLastName());
        System.out.println("sex "+st2.getSex());
        System.out.println("Department Name:"+st2.getDepartment());
        System.out.println("Birth Date:"+st2.getBirtDate());
    }
}
```

Calling and returning methods

- To use a method, you have to call or invoke it.
- There are two ways to call a method; the choice is based on whether the method returns a value or not.
- If the method returns a value, a call to the method is usually treated as a value. For example,

```
int larger = max(3, 4);
```

calls max(3, 4) and assigns the result of the method to the variable larger.

- Another example of a call that is treated as a value is

```
System.out.println(max(3, 4));
```

which prints the return value of the method call max(3, 4).

Calling and returning methods Cont.

- If the method returns void, a call to the method must be a statement. For example, the method println returns void.
- The following call is a statement:

```
System.out.println("Welcome to Java!");
```

- This program contains the main method and the max method.
- The main method is just like any other method except that it is invoked by the JVM.

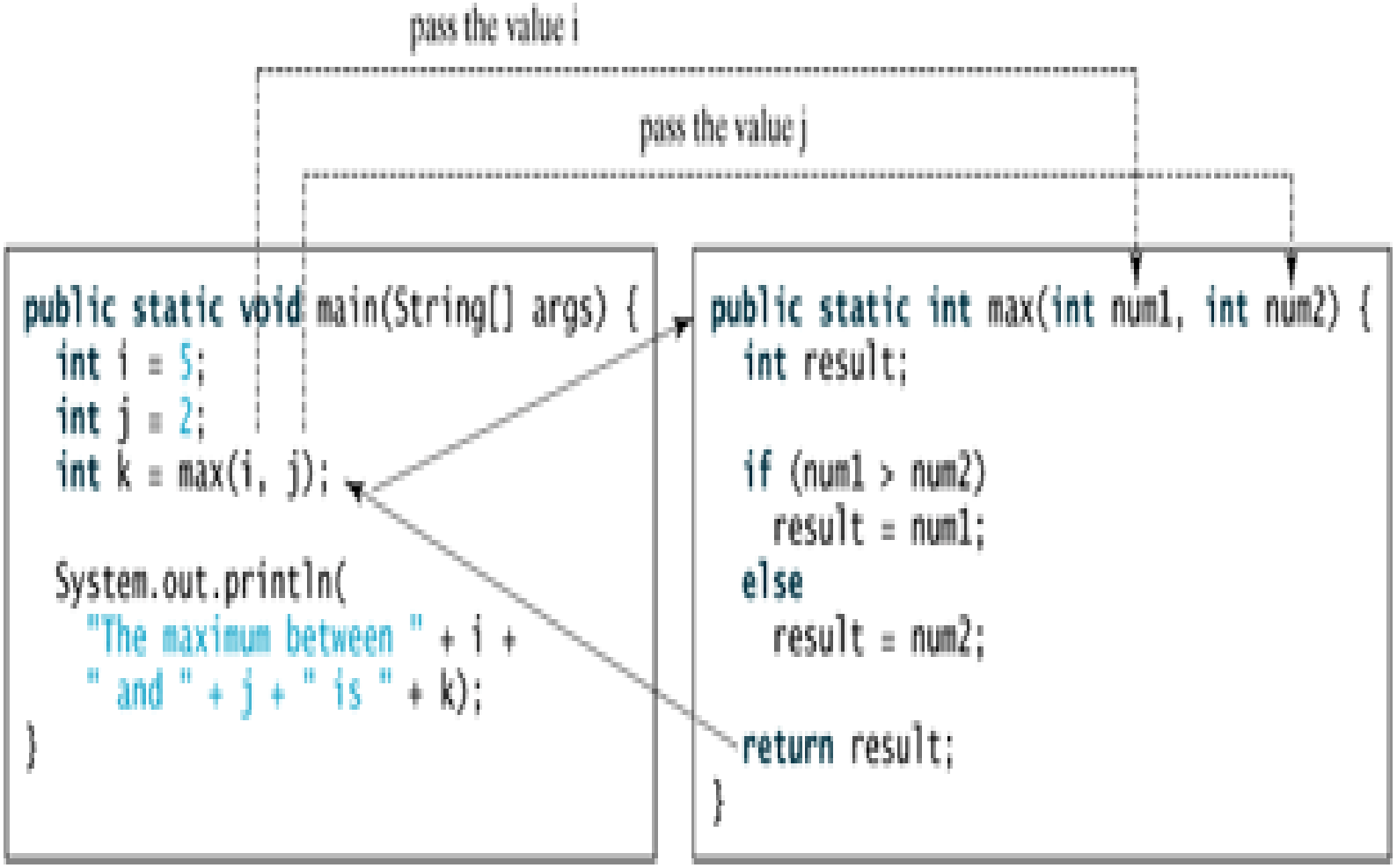
Calling and returning methods Cont.

```
 1  public class TestMax {
 2    /** Main method */
 3    public static void main(String[] args) {
 4      int i = 5;
 5      int j = 2;
 6      int k = max(i, j);
 7      System.out.println("The maximum between " + i +
 8        " and " + j + " is " + k);
 9    }
10
11    /** Return the max between two numbers */
12    public static int max(int num1, int num2) {
13      int result;
14
15      if (num1 > num2)
16        result = num1;
17      else
18        result = num2;
19
20      return result;
21    }
22  }
```

Calling and returning methods Cont.

- When the max method is invoked (line 6), variable i's value 5 is passed to num1, and variable j's value 2 is passed to num2 in the max method.
- The flow of control transfers to the max method.
- The max method is executed.
- When the return statement in the max method is executed, the max method returns the control to its caller (in this case the caller is the main method).
- This process is illustrated in below.
  - **When the max method is invoked, the flow of control transfers to the max method. Once the max method is finished, it returns the control back to the**

Calling and returning methods Cont.



Static/class Variables

- Class variables are variables that are defined and stored in the class itself. Their values, therefore, apply to the class and to all its instances.

- With instance variables, each new instance of the class gets a new copy of the instance variables that class defines. Each instance can then change the values of those instance variables without affecting any other instances.

- With class variables, there is only one copy of that variable. Every instance of the class has access to that variable, but there is only one value. Changing the value of that variable changes it for all the instances of that class.

Static/class Variables Cont.

- You define class variables by including the static keyword before the variable itself. For example, take the following partial class definition:

```
class FamilyMember {
static String fatherNme = "Jemal";
String name;

…
}
```

- Instances of the class FamilyMember each have their own values for name. But the class variable fatherName has only one value for all family members. Change fatherName, and all the instances of FamilyMember are affected.

Static/class Variables Cont.

- To access class variables, you use the same dot notation as you do with instance variables.

- To get or change the value of the class variable, you can use either the instance or the name of the class on the left side of the dot. Both of the lines of output in this example print the same value:

```
FamilyMember dad = new FamilyMember();

System.out.println("Family's fatherName is: " + dad. fatherName );

System.out.println("Family's fatherName is: + FamilyMember. fatherName);
```

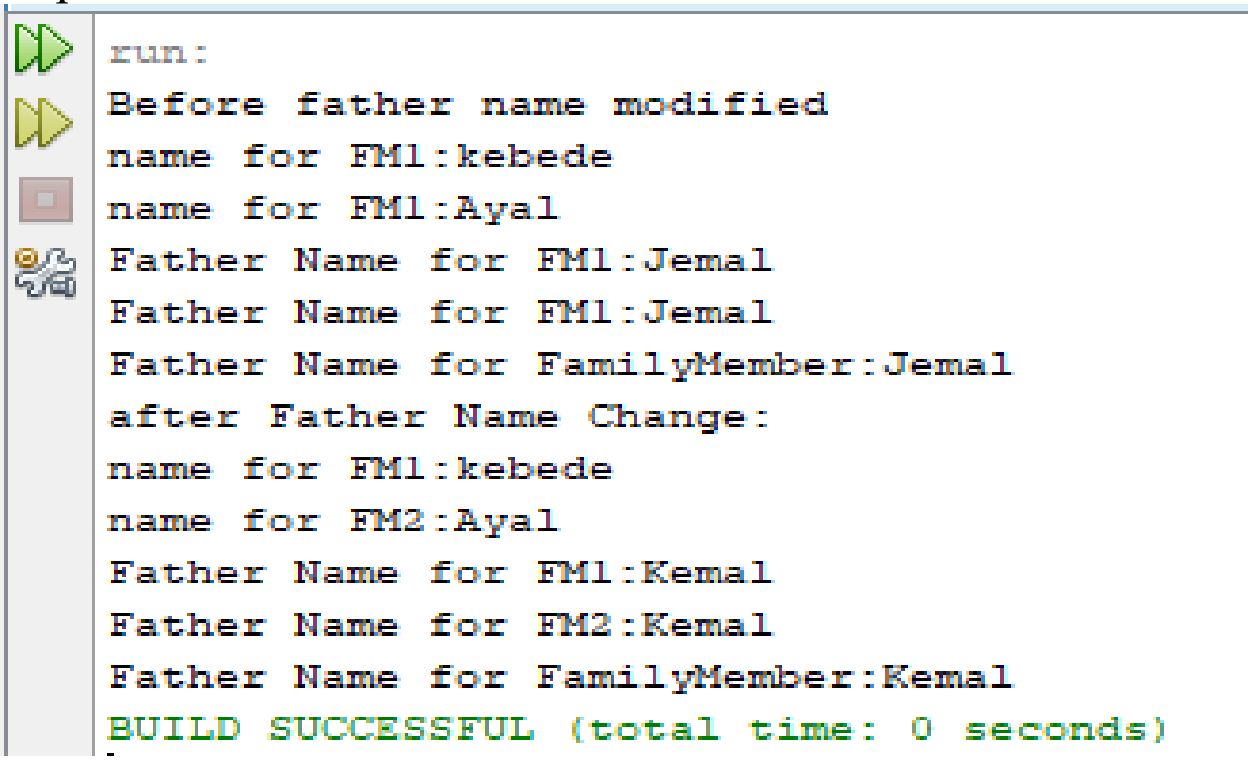
Static/class Variables Cont.

```
1  package staticexample;
2  public class FamilyMember {
3      static String FatherName="Jemal";
4      String name;
5      public static void main(String[] args) {
6          FamilyMember FM1=new FamilyMember();
7          FM1.name="kebede";
8          FamilyMember FM2=new FamilyMember();
9          FM2.name="Ayal";
10         System.out.println("Before father name modified");
11         System.out.println("name for FM1:"+FM1.name);
12         System.out.println("name for FM1:"+FM2.name);
           System.out.println("Father Name for FM1:"+FM1.FatherName);
           System.out.println("Father Name for FM1:"+FM2.FatherName);
15         System.out.println("Father Name for FamilyMember:"+FamilyMember.FatherName);
           FM1.FatherName="Kemal";//or FM1.FatherName="Kemal" or FamilyMember.FatherName="Kemal"
17         System.out.println("after Father Name Change:");
18         System.out.println("name for FM1:"+FM1.name);
19         System.out.println("name for FM2:"+FM2.name);
           System.out.println("Father Name for FM1:"+FM1.FatherName);
           System.out.println("Father Name for FM2:"+FM2.FatherName);
22         System.out.println("Father Name for FamilyMember:"+FamilyMember.FatherName);
23
24         }
25 }
```

The same but you should be recommended using class name for static object

Static/class Variables Cont.

- Output



```
run:
Before father name modified
name for FM1:kebede
name for FM1:Ayal
Father Name for FM1:Jemal
Father Name for FM1:Jemal
Father Name for FamilyMember:Jemal
after Father Name Change:
name for FM1:kebede
name for FM2:Ayal
Father Name for FM1:Kemal
Father Name for FM2:Kemal
Father Name for FamilyMember:Kemal
BUILD SUCCESSFUL (total time: 0 seconds)
```