Object Oriented Programming

Introduction

- An object is a particular and finite element in a larger model.
- A class is a group of objects that have something in common.
- A class type variable contains a “handle” (reference) for a specific object.

Class Definition

```
[access_modifier] class  name_of_the_class
{
    [variables declaration]
    [methods declaration]
}
```

Class Definition

```
class Rectangle
{
    double width;
    double height;
    double area()
    {
          return width*height;
    }
    …
}
```

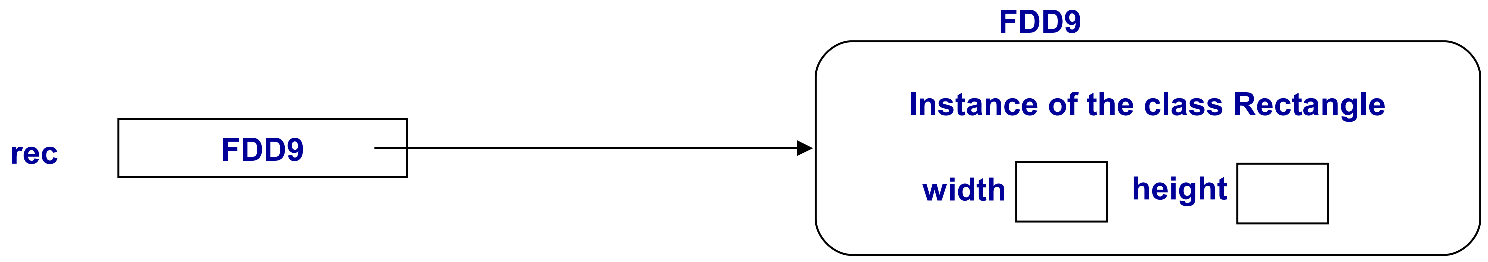
Class Type Variable

- The class type variable isn’t an object. It is a simple variable that can hold a reference for a specific object.
- When the class type variable holds a reference of a specific object we can use that variable in order to access the variables inside the object as well as for calling various methods on it.

Class Type Variable

- Given the class `Rectangle`, and a class type variable named `rec`, initializing the `rec` variable might look the following:

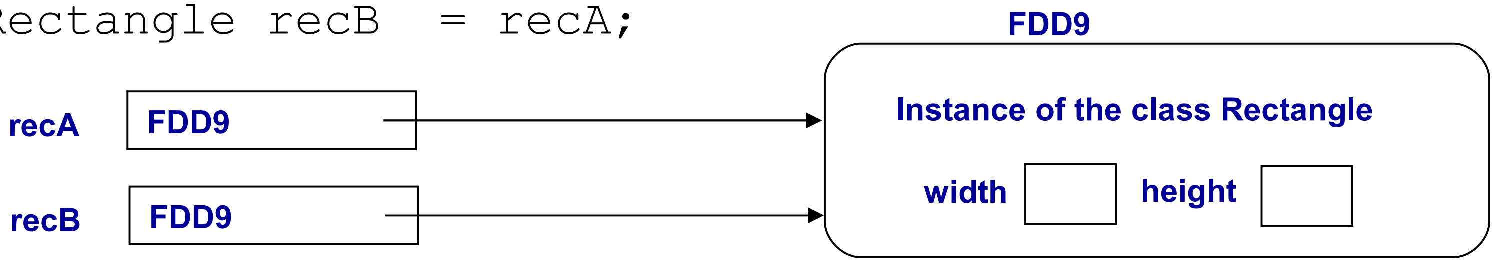
```
Rectangle rec = new Rectangle();
```



Class Type Variable

Given the class `Rectangle`, and two class type variables named `recA` and `recB`, the following code results in two variables that point to the same object.

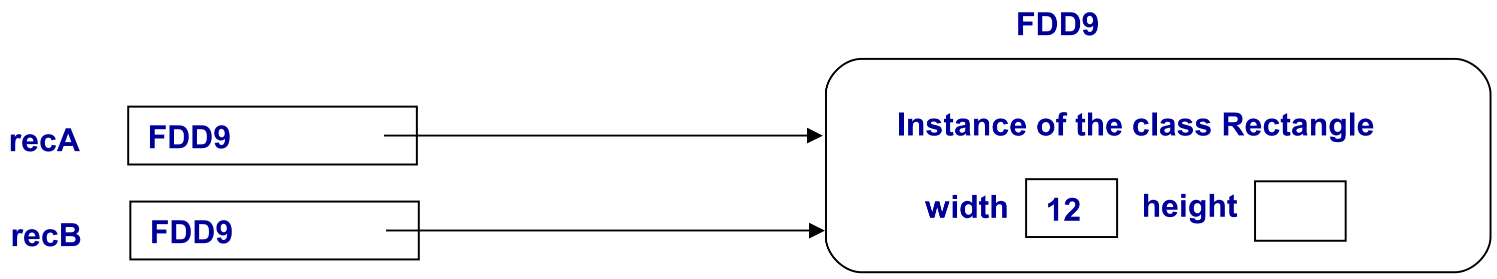
```
Rectangle recA = new Rectangle();
Rectangle recB  = recA;
```



Class Type Variable

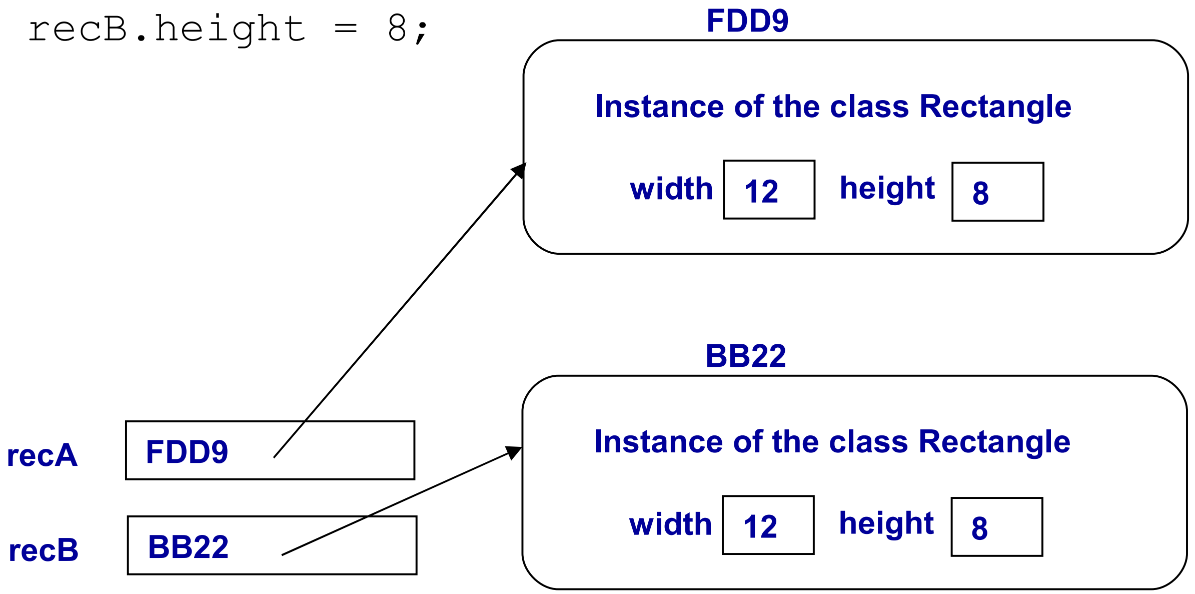
- We can easily use the dot (.) in order to access the object variables, get their values and set new ones.

```
Rectangle recA = new Rectangle();
Rectangle recB = recA;
```



Class Type Variable

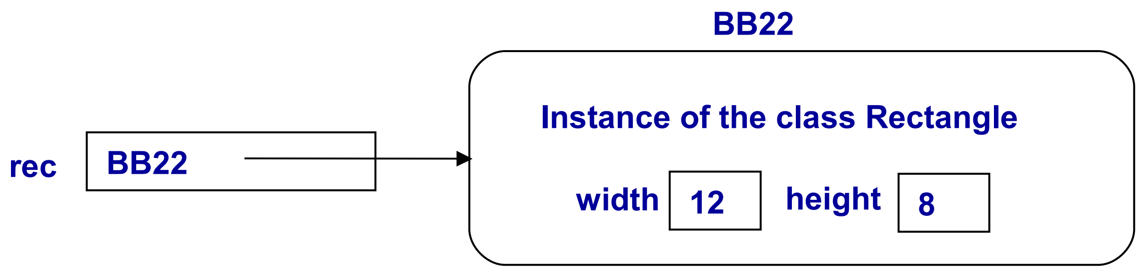
```
Rectangle recA = new Rectangle();
Rectangle recB = new Rectangle();
recA.width = 12; recB.width = 12;
recA.height = 8; recB.height = 8;
if(recA==recB)
{
      ...
}
```


FDD9
Instance of the class Rectangle
width 12 height 8
BB22
Instance of the class Rectangle
width 12 height 8
recA FDD9
recB BB22

Class Type Variable

- We can easily use the dot(.) in order to invoke a method on the object.

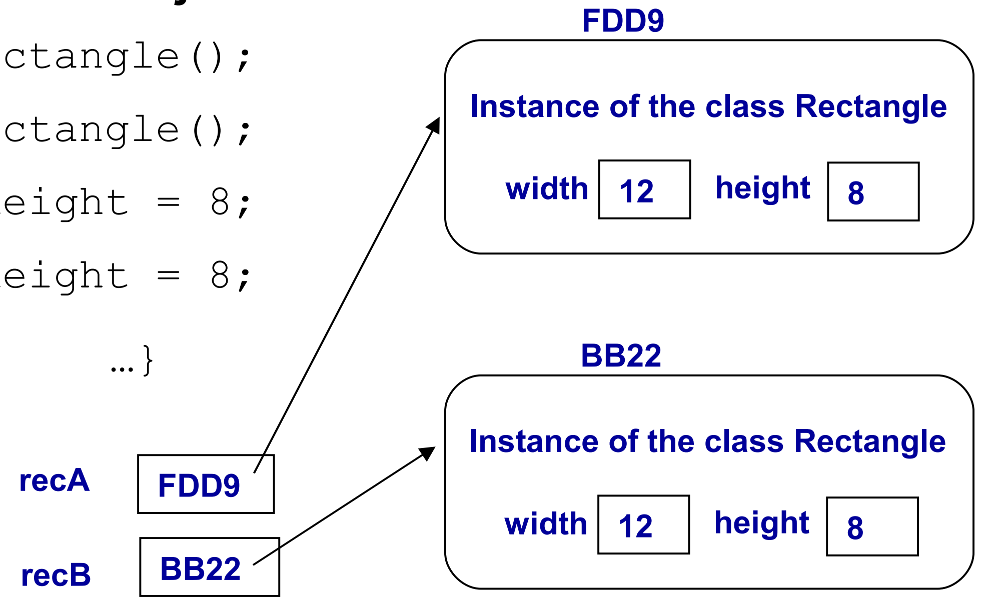
```
Rectangle rec = new Rectangle();
rec.width = 12;
rec.height = 8;
System.out.println("area is "+rec.area());
```



Class Type Variable

- Calling the `equals` method is the right way for comparing between two objects

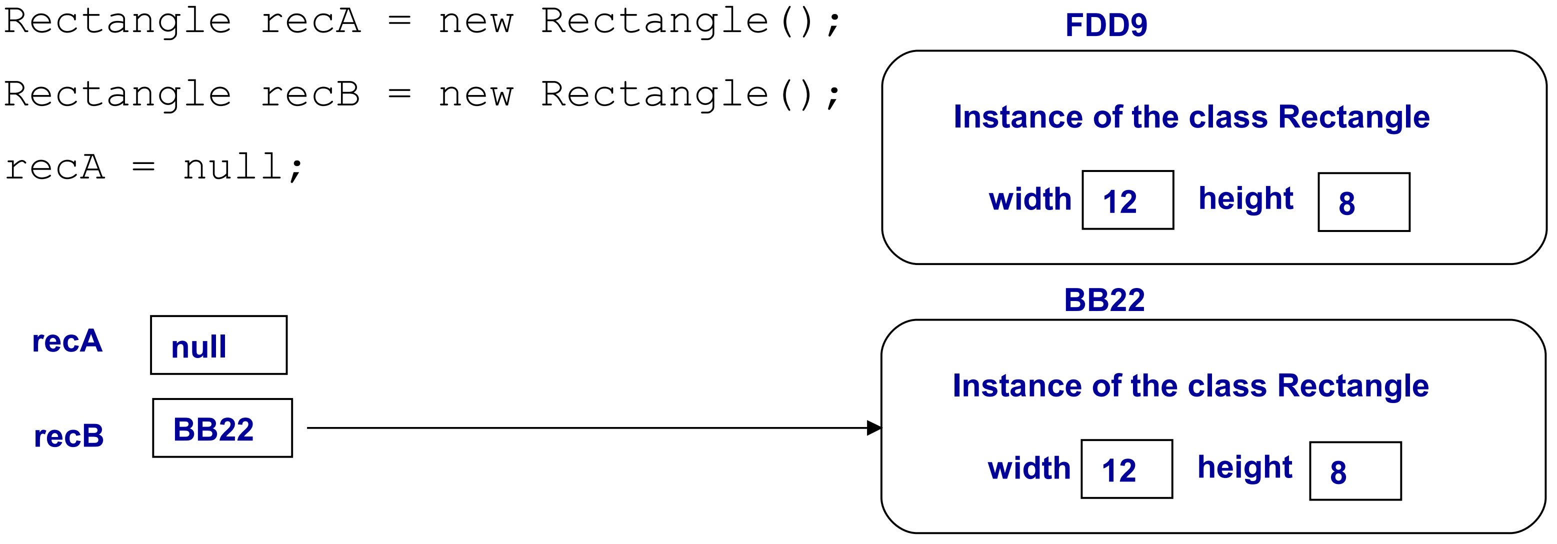
```
Rectangle recA = new Rectangle();
Rectangle recB = new Rectangle();
recA.width = 12; recA.height = 8;
recB.width = 12; recB.height = 8;
if(recA.equals(recB)) {     …}
```


FDD9
Instance of the class Rectangle
width
12
height
8
BB22
Instance of the class Rectangle
width
12
height
8
recA
FDD9
recB
BB22

Class Type Variable

- The special value `null` can be assigned to any class type variable.

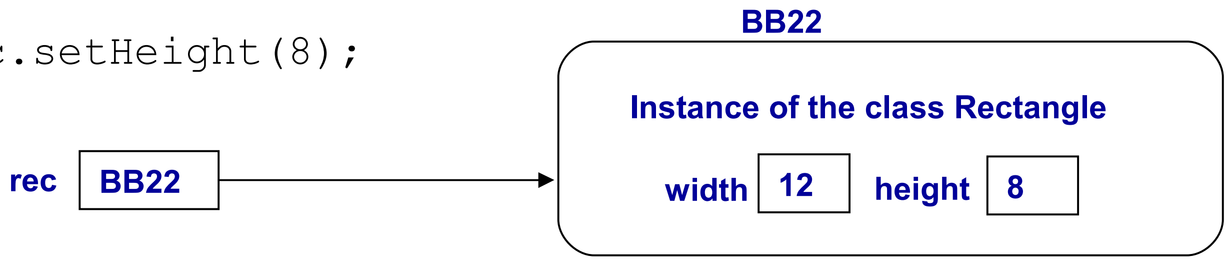
```
Rectangle recA = new Rectangle();
Rectangle recB = new Rectangle();
recA = null;
```


FDD9
Instance of the class Rectangle
width 12 height 8
BB22
Instance of the class Rectangle
width 12 height 8
recA null
recB BB22

Calling Methods

- We can call a method on a specific object by writing the reference for that object following with a dot (.) and the name of the method right after it.

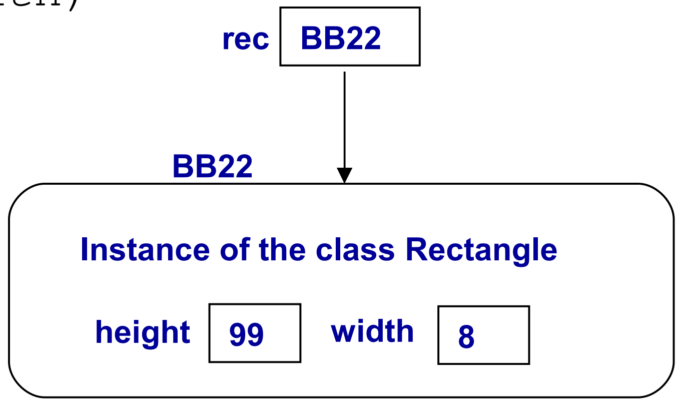
```
Rectangle rec = new Rectangle();
rec.setWidth(12);
rec.setHeight(8);
```



The `this` Keyword

- The `this` keyword holds the reference of the current object. We can use it within instance methods or constructors only.

```
void setWidth(double width)
{
    this.width = width;
}
```



Class Definition

```
[access_modifier] class  <class_name>

{

   <attributes declaration>

   <constructors declaration>

   <methods declaration>

   ...

}
```

Fields Definition

```
[modifiers] <type> <name> [=default value];
```

Methods Definition

```
[modifiers] <return_type> <name> (
       <parameter_type> <parameter_name>, ..)
{
  <statements>
}
```

Methods Overloading

- The same method can be defined in the same class in several different versions.
- All versions should be defined with a returned value of the same type.
- The parameters list in all method must be with a different signature.

Methods Overloading

```
public void doSomething() {…}

public void doSomething(int width) {…}

public void doSomething(int width, int height) {…}`1
```

Constructors Definition

```
[modifier] <class_name> (<parameter_type> <parameter
   name>, ..)
{
   <statements>
}
```

Default Constructor

- The default constructor exists when we define a class without any constructor.
- The default constructor has no parameters and it doesn’t have a body.

Overloading Constructors

- Constructors can be overloaded (like methods).

```
public Rectangle() {...}
public Rectangle(int num)  {...}
public Rectangle(int num1, int num2)  {...}
```

- Each constructor should be defined with a different signature.

Using `this` Within Constructors

❖ Placing the `this` keyword in the first line (within the constructor block) we can call another constructor.

Using `this` within constructors

```
class Rectangle
{
   int width, height;
   Rectangle()
  {
        this(2,3);
  }
   Rectangle(int num1, int num2)
   {
        width = num1;
        height = num2;
   }
}
```

Static Variables

- Class variable that was defined together with the `static` keyword as one of its modifiers is called a 'static variable' (AKA 'class variable').
- The `static` keyword associates the variable with the class as a whole rather than with a particular instance.

Static Variables

- The static variable can be accessed from within any method of the class.
- The static variable can also be accessed from outside of the class scope if its access modifier allows it.
- Static variables can be accessed by using a class type reference or by using the name of the class.

Static Variables

```
public static void main(String args[])
{
     Rectangle myRectangle, hisRectangle;
     myRectangle = new Rectangle();
     hisRectangle = new Rectangle(“Juka”);
     System.out.println(
       “The number of Rectangles instances is : “
       + Rectangle.numOfRectangles);
}
```

Static Methods

- Method that was defined together with the `static` keyword as one of its modifiers is called a 'static method' (AKA 'class method').
- The ‘`static`’ keyword as a method modifier associates the method with the class as a whole rather than with a particular instance.

Static Initializers

- A ‘static block’ is a block of code that doesn’t belong to any specific method. The ‘static block’ is prefixed with the key word ‘`static`’.
- The ‘static block’ contains code which is executed when the class is loaded to the JVM memory.
- The code within the ‘static block’ is executed only once (when the class is loaded).

Final Variables

- A final variable can be set only once. The final variable assignment can occur independently of its declaration.
- A final variable that its value wasn’t set together with its declaration must be set in every constructor.

Software Packages

- Package is a group of classes and interfaces.
- Packages help manage large software applications.
- Package can contain sub packages.

The Package Statement

- The package statement starts with the word `package`.

```
package <package_name>.<package_name>….  ;
```

The following is an example for declaring a package with an hierarchy of three levels.

```
package com.zozobra.javaProjects;
```

The Package Statement

- Only one package statement per one source file is allowed.
- If the package statement isn’t included within the source file then all classes in that specific source file will be belonged of the default package.

The import statement

- The import statement can import either a specific class or all classes (that belong to specific package).

```
import <package name>.<package name>.<class name>;
import <package name>.<package name>.*;
```

- The following example imports all classes that belong to the java.awt package.

```
import java.awt.*;
```

The Java API Documentation

- The Java API Documentation is a set of HTML documents that provide information about the Java API.
- The API Documentation provides information regarding each of the packages and each one of their classes and interfaces.

Class Design

- The variables you declare should always be private unless there is a good reason not doing so.
- Name the classes and their variables and methods with meaningful names that reflect their responsibilities.
- Break up classes to few more little classes when needed.
- Use a standard format for class definition.

Enums

- In the past, enumerated types were represented using the following pattern:

```
...
public final static int JANUARY = 1;
public final static int FEBRUARY = 2;
...
```

Enums

- The enumerated types support presented in Java SE 5.0 in their simplest form look like C/C++ enums.

```
public enum Months {JANUARY, FEBRUARY, MARCH};
```

Enums

```
enum Months {  JAN, FEB, MAR, APR, MAY, JUN, JUL,
                    AUG, SEP, OCT, NOV, DEC};
enum Seasons {WINTER,SPRING,SUMMER,AUTOMN};

class SimpleEnumSample
{
    public static void main(String[] args)
    {
        System.out.println(Seasons.WINTER);
        System.out.println(Months.JAN);
        System.out.println(Months.FEB);
        System.out.println(Months.MAR);
    }
}
```

Enums

- Declaring an enum means declaring a new full fledged class. As such, it is possible to add methods and fields, implement interfaces and more.

Enums

```
public enum Currencies
{
    USD,EURO,GBP,SGD;

    private double exchangeRate;

    public void setExchangeRate(double value)

    {

        exchangeRate = value;

    }

    public double convert(double sum, Currencies otherCurrency)
    {

        return sum / this.exchangeRate /
          otherCurrency.exchangeRate;

    }

}
```

Enums

```
public class CurrenciesDemo
{
    public static void main(String[] args)
    {
        Currencies currA, currB;
        currA = Currencies.EURO;
        currB = Currencies.GBP;
        currA.setExchangeRate(0.8);
        currB.setExchangeRate(0.4);
        double sum = currA.convert(120, Currencies.GBP);
        System.out.println("EURO 120 = GBP " + sum);
    }
}
```

Enums

- The enum types provide high-quality implementations of all the Object methods (e.g. `toString()`).
- All enum types are both `Comparable` and `Serializable`.

Enums

- Each enum type has the static `values()` method that returns a reference for an array that holds references for the objects that represent the Enum possible values.

  In other words, the array holds all of the values the enum type includes. The order is the same order in which these values were added to this enum type.

Enums

```
public enum Currencies {USD,EURO,GBP,SGD;}

public class CurrenciesPossibleValues
{
    public static void main(String[] args)
    {
        Currencies vec[] = Currencies.values();
        for(Currencies ob : vec)
        {
            System.out.println(ob);
        }
    }
}
```

Enums

- Each one of the enum possible values can be initialized by adding a brackets with the values to use. A compatible private constructor should be defined.

Enums

```
public enum LengthUnits
{
    cm(1),kilometer(100000),meter(100),mile(160934.4),feet(30.48);

    private double conversionRateToCm;
    private LengthUnits(double conversionRateToCm)
    {
        this.conversionRateToCm = conversionRateToCm;
    }
}
```

Enums

- Within the enum type it is possible to define an abstract method and override it by defining a concrete method in each one of the enum defined constants.

  This way, each enum constant can get a different behavior for the same method. Declaring a concrete method in a specific constant is done by declaring the method within brackets that follow the constant declaration.

Enums

```
enum Operator
{
  PLUS
    {double operate(double num1, double num2)
        {return num1+num2;}},
  MINUS
    {double operate(double num1, double num2)
        {return num1-num2;}};
  abstract double operate(double num1, double num2);
}
```

Enums

- The java.util package includes two classes that are very useful to use when dealing with enum types.

  The EnumSet Class

  Implements the Set interface. This is a high performance Set implementation for enums. When using the EnumSet, all members must be of the same enum type.

Enums

```
enum Day { SUN, MON, TUE, WED, THU, FRI, SAT };
EnumSet<Day> subSet = EnumSet.range(Day.SUN, Day.THU);
for(Day day : subSet)
{
     System.out.println(day);
}
```

Enums

The EnumMap Class

Implements the Map interface. This is a high performance Map implementation for enums. The keys should be of the same enum type.

Using the following constructor it is possible to create a new EnumMap that its keys are of a specific enum type:

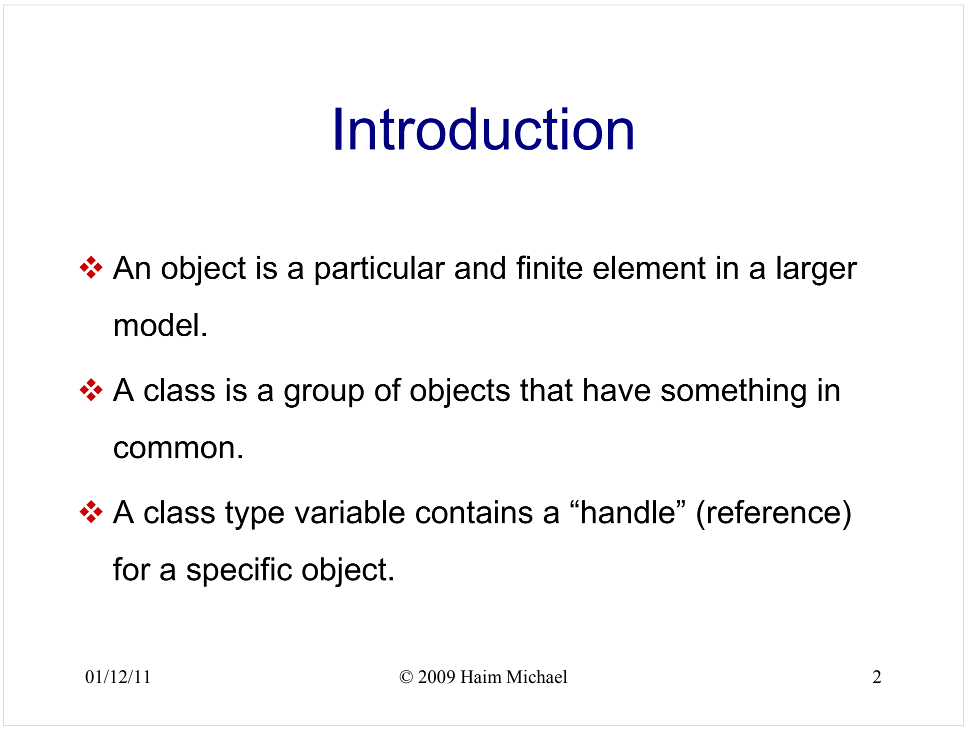
```
public EnumMap(Class<K> keyType)
```

Enums

```
enum Day { SUN, MON, TUE, WED, THU, FRI, SAT };
EnumMap<Day,String> map =
            new EnumMap<Day,String>(Day.class);
map.put(Day.SUN,”Sunday”);
map.put(Day.MON,”Monday”);
System.out.println(map);
```

Object Oriented Programming

01/12/11 © 2009 Haim Michael 1



Introduction

- An object is a particular and finite element in a larger model.
- A class is a group of objects that have something in common.
- A class type variable contains a “handle” (reference) for a specific object.

01/12/11 © 2009 Haim Michael 2

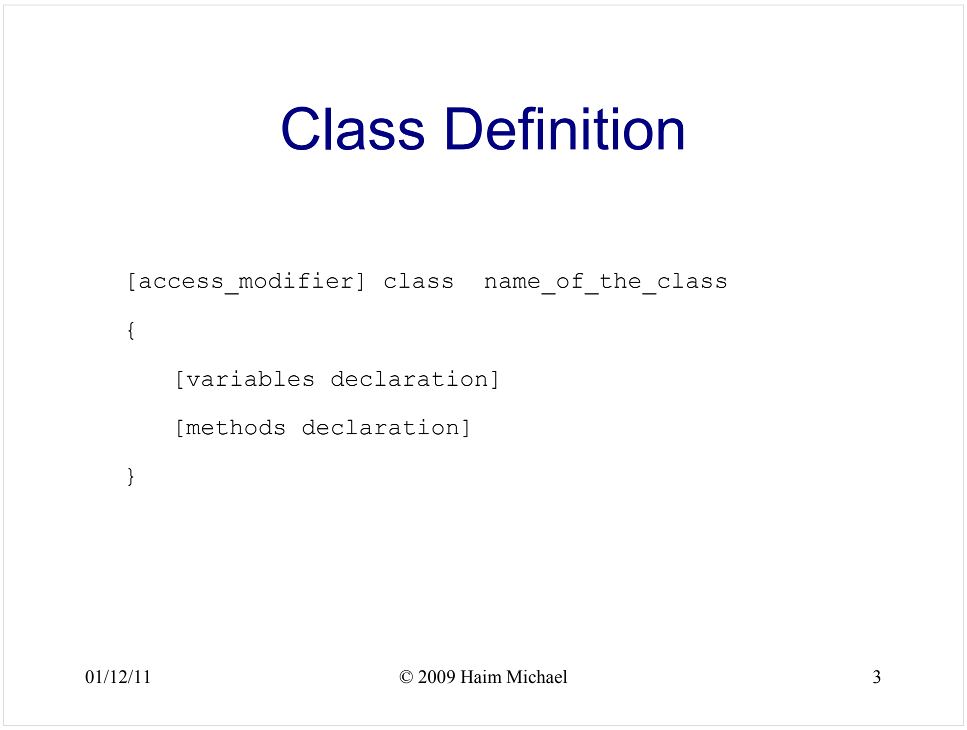
An object is a particular and finite element in a larger model. An object may be very concrete, such as a particular book in a library. An object may be invisible and conceptual, such as a meeting between a boy and a girl. An object may have a short life, such as a book lending in the library.

The abstraction comprising books in a library includes the name of the book, the name of the author, the book edition etc… Each book in the library has its own specific name, author name and edition. Each object (book) has its state, which describes its characteristics and current condition. Some characteristics, such as the book name and the author name never change and some of the characteristics do change (like the number of readers that lended the book).

Objects also have behavior, which defines the actions that other objects may perform on the object. For instance, a student(another object) can lend a book from the library. Such action can result in invoking the lend method on that book (object).

The work in an object-oriented system is divided up among many objects. Each object is configured for its particular role in the system. Since each object has a fairly narrow set of responsibilities, the objects must cooperate to accomplish larger goals.

A method is a service or responsibility that an object exposes to other objects. Thus, one object can call another object’s methods. A method is loosely analogous to a function or subroutine in procedural programming, except that the method is called on a specific object that has its own state.



Class Definition

```
[access_modifier] class  name_of_the_class
{
    [variables declaration]
    [methods declaration]
}
```

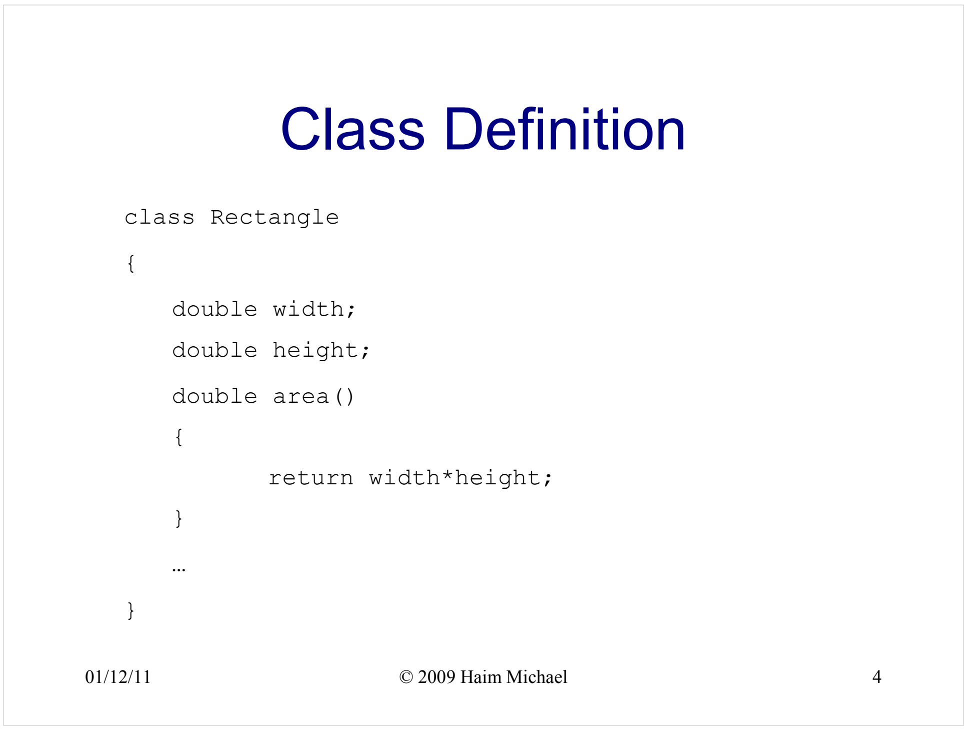
01/12/11 © 2009 Haim Michael 3

The class is like a template. When declaring a class, each new instance of that class will have within it each of the variables that were declared. Further more, it will be possible invoking each one of the methods on each one of the class instances.

Later, the possibilities of declaring static variables and static methods will be presented. Static variables and static methods have a different meaning.

An example for a class declaration:

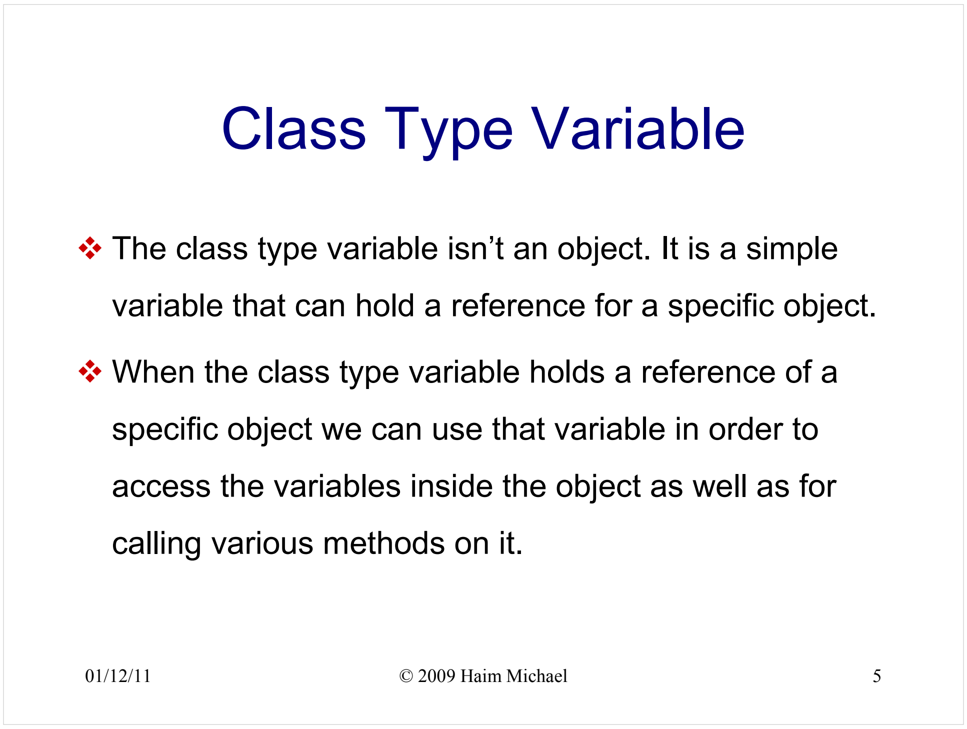
```
class Student
{
            private String name;
            private int age;
            public void setAge(int ageVal)
            {
                        age = ageVal;
            }
}
```



Class Definition

```
class Rectangle
{
    double width;
    double height;
    double area()
    {
           return width*height;
    }
    …
}
```

The access modifier can be either public or none, which means package friendly. The meaning of these options will be explained later.



Class Type Variable

- The class type variable isn't an object. It is a simple variable that can hold a reference for a specific object.
- When the class type variable holds a reference of a specific object we can use that variable in order to access the variables inside the object as well as for calling various methods on it.

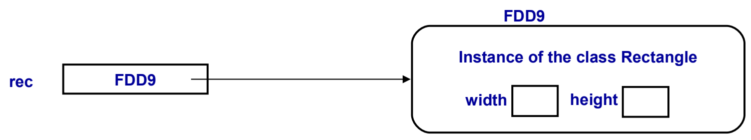
01/12/11 © 2009 Haim Michael 5

Unlike in C\C++, the class type variable isn't an object. Trying referring the object variables or invoking a method using a class type variable that holds null will result in a NullPointerException.

Class Type Variable

- Given the class `Rectangle`, and a class type variable named `rec`, initializing the `rec` variable might look the following:

```
Rectangle rec = new Rectangle();
```



ven though the reference isn’t the real address of the object you might treat it as if it was. That might help you understanding the material. However, you should remember that the real address is known only to the JVM that handles the memory independently.

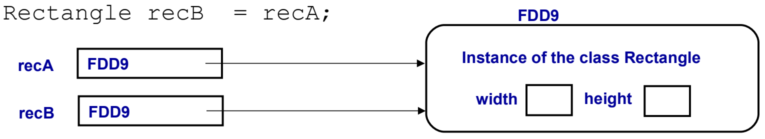
The above scheme assumes that the object reference is FDD9.

The scheme assumes that the class Abc has two instance variables: var1 and var2.

Class Type Variable

Given the class `Rectangle`, and two class type variables named `recA` and `recB`, the following code results in two variables that point to the same object.

```
Rectangle recA = new Rectangle();
Rectangle recB  = recA;
```



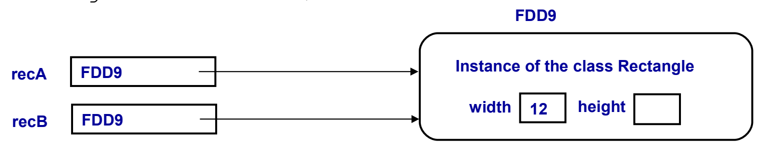
Changing the object can be done using myAbc or hisAbc. The effect will be the same.

The following slides assume the existence of the class Abc and the class type variables: myAbc and hisAbc.

Class Type Variable

- We can easily use the dot (.) in order to access the object variables, get their values and set new ones.

```
Rectangle recA = new Rectangle();
Rectangle recB = recA;
```



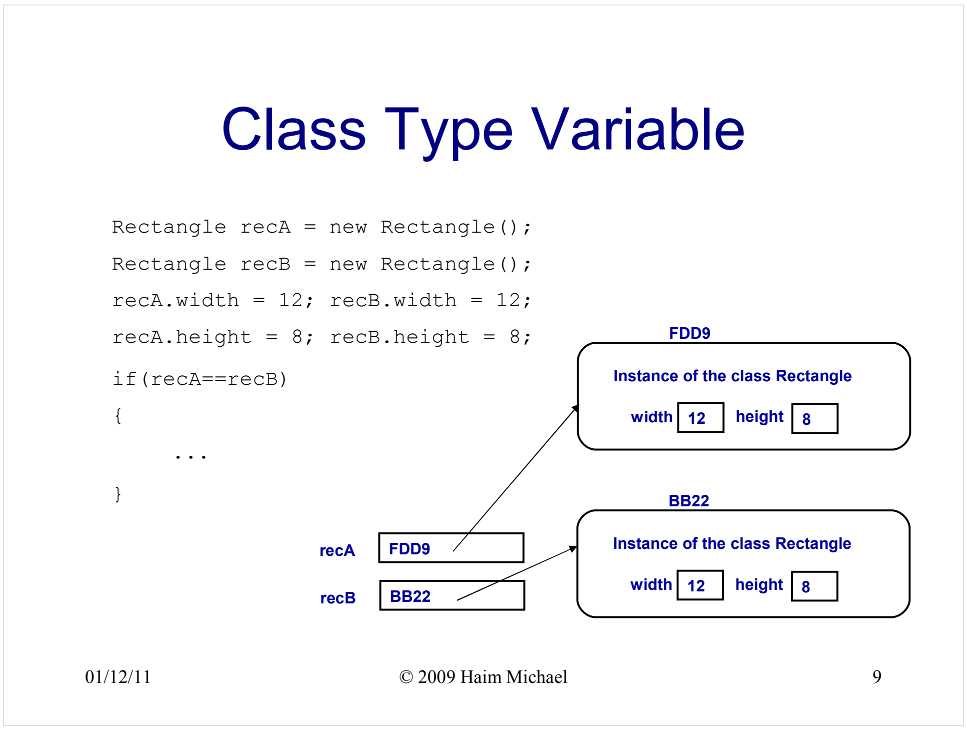
01/12/11 © 2009 Haim Michael 8

Given this code, the output of each one of the following code segments will be the same,

```
System.out.println(myAbc.var1);
```

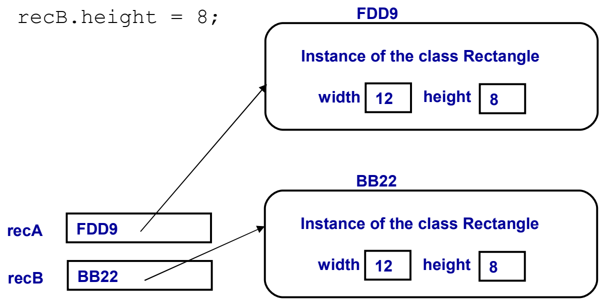
or

```
System.out.println(hisAbc.var1);
```

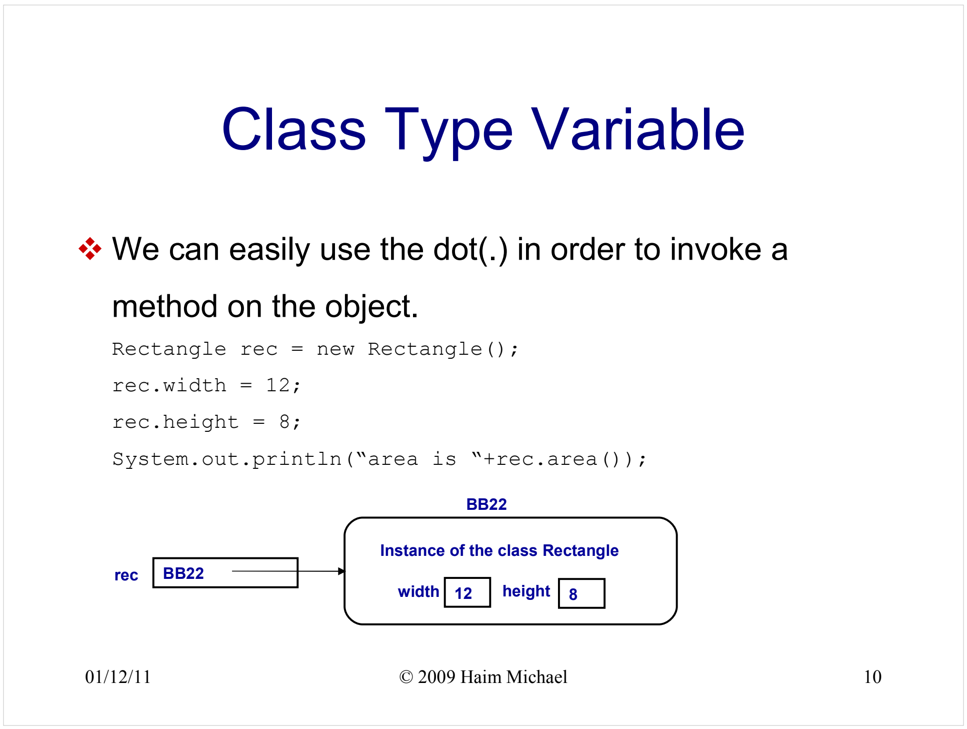


Class Type Variable

```
Rectangle recA = new Rectangle();
Rectangle recB = new Rectangle();
recA.width = 12; recB.width = 12;
recA.height = 8; recB.height = 8;
if(recA==recB)
{
     ...
}
```


FDD9
Instance of the class Rectangle
width 12 height 8
BB22
Instance of the class Rectangle
width 12 height 8
recA FDD9
recB BB22

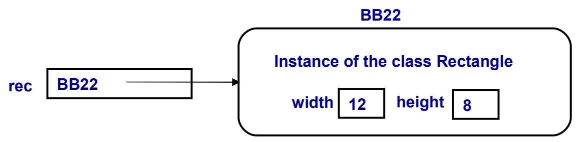
Even though the two objects are the same, they have different references. Therefore, comparing them gives false.



Class Type Variable

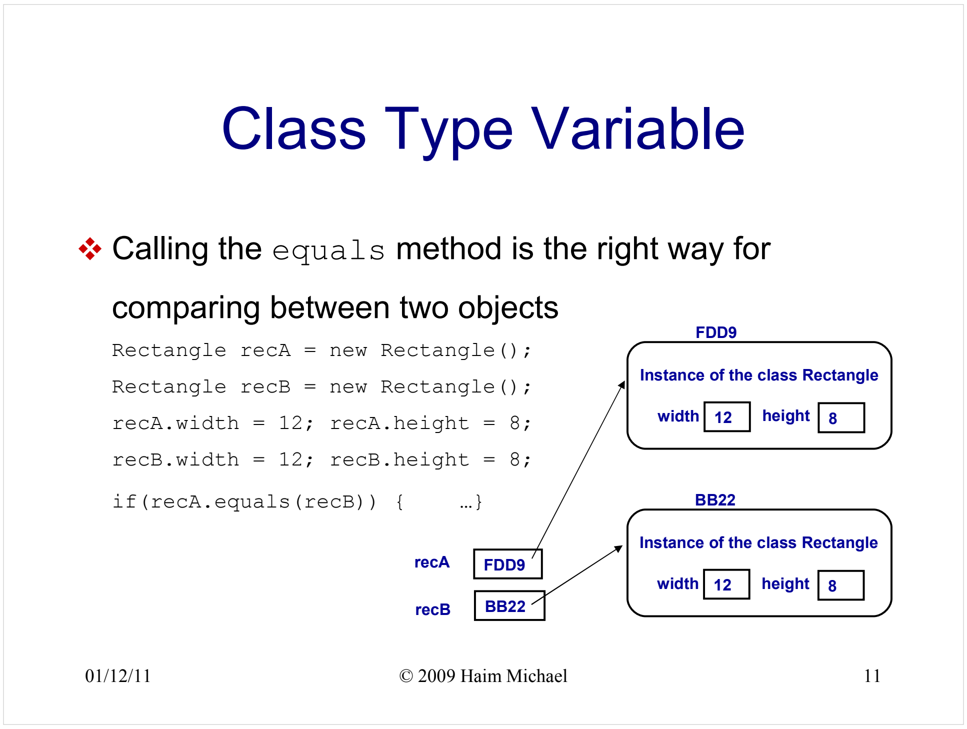
❖ We can easily use the dot(.) in order to invoke a method on the object.

```
Rectangle rec = new Rectangle();
rec.width = 12;
rec.height = 8;
System.out.println("area is "+rec.area());
```



```
Given that Xxx class declaration looks as follows:
class Xxx
{
            int var1;
            void getDetails()
            {
                        System.out.println("var1="+var1);
            }
}

The output will be:
Var1=12
```

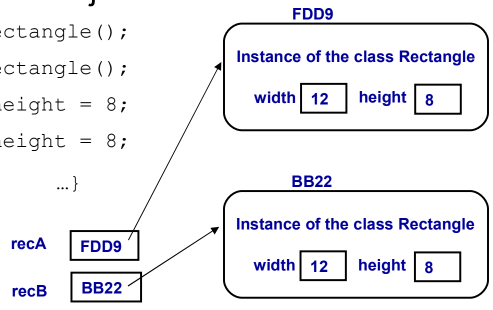
When a method is invoked on an object it can access its variables and even change their values.



Class Type Variable

❖ Calling the `equals` method is the right way for comparing between two objects

```
Rectangle recA = new Rectangle();
Rectangle recB = new Rectangle();
recA.width = 12; recA.height = 8;
recB.width = 12; recB.height = 8;
if(recA.equals(recB)) {    …}
```


FDD9
Instance of the class Rectangle
width 12 height 8
BB22
Instance of the class Rectangle
width 12 height 8
recA FDD9
recB BB22

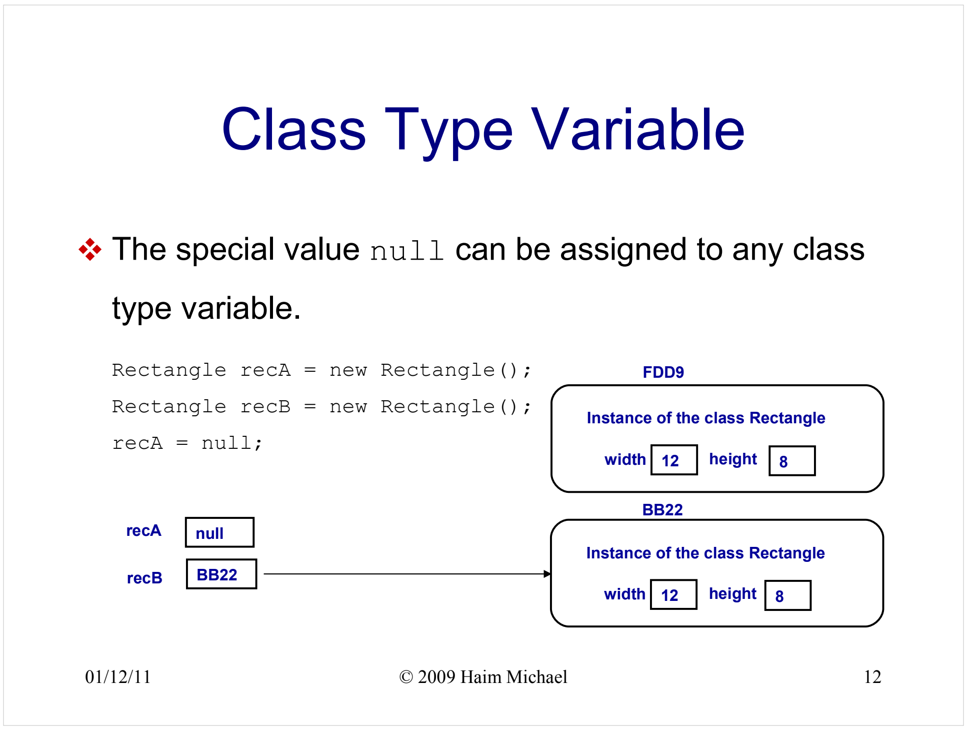
01/12/11 © 2009 Haim Michael 11

The equals method can be invoked on every object since it was declared in the Object class. Every existing class extends Object (directly or indirectly) and therefore the equals method can be invoked on every object. All is needed, is overriding the equals method inside the Abc class so it shall have a real meaning.

If you aren't familiar with the inheritance mechanism you can return to this slide after finishing the Inheritance module.

The Abc class might look as follows:

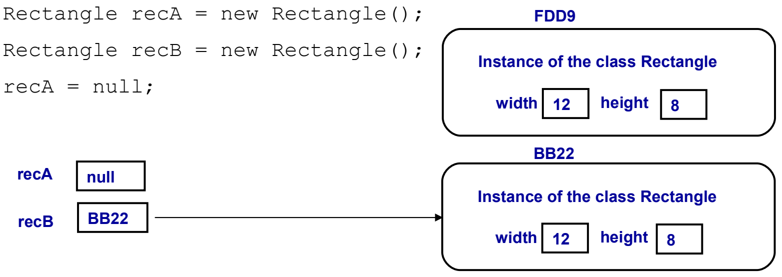
```
class Abc
{
  int var1, var2;

  public boolean equals(Abc otherAbc)
  {
    return (var1==otherAbc.var1 && var2==otherAbc.var2);
  }
}
```



Class Type Variable

- The special value `null` can be assigned to any class type variable.

```
Rectangle recA = new Rectangle();
Rectangle recB = new Rectangle();
recA = null;
```

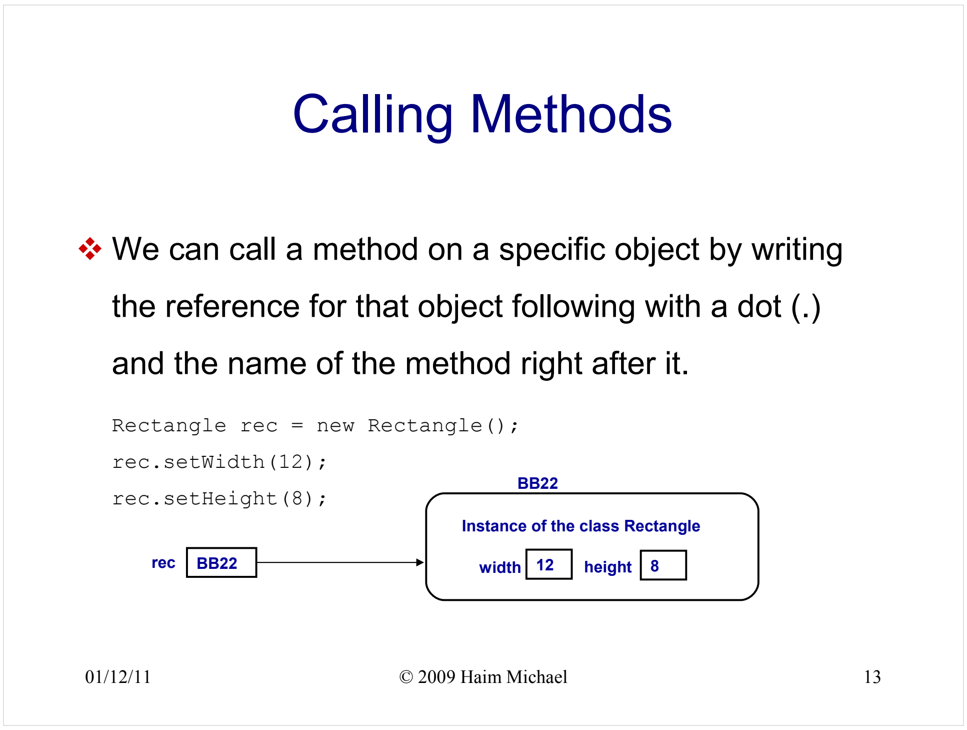

FDD9
Instance of the class Rectangle
width 12 height 8
BB22
Instance of the class Rectangle
width 12 height 8
recA null
recB BB22

01/12/11 © 2009 Haim Michael 12

The special value ‘null’ is not a constant that equals to 0 like in C\C++. The ‘null’ is a keyword and not a constant. Since Java is a case sensitive language, you should write ‘null’ and not ‘NULL’ !!!

One of the common techniques making the garbage collector freeing the allocated memory of a specific object is placing ‘null’ in each one of variables that holds the object reference.

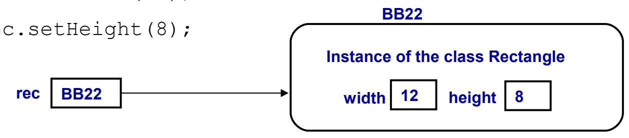
In the above example, the variable myAbc doesn’t hold an object’s references.



Calling Methods

❖ We can call a method on a specific object by writing the reference for that object following with a dot (.) and the name of the method right after it.

```
Rectangle rec = new Rectangle();
rec.setWidth(12);
rec.setHeight(8);
```

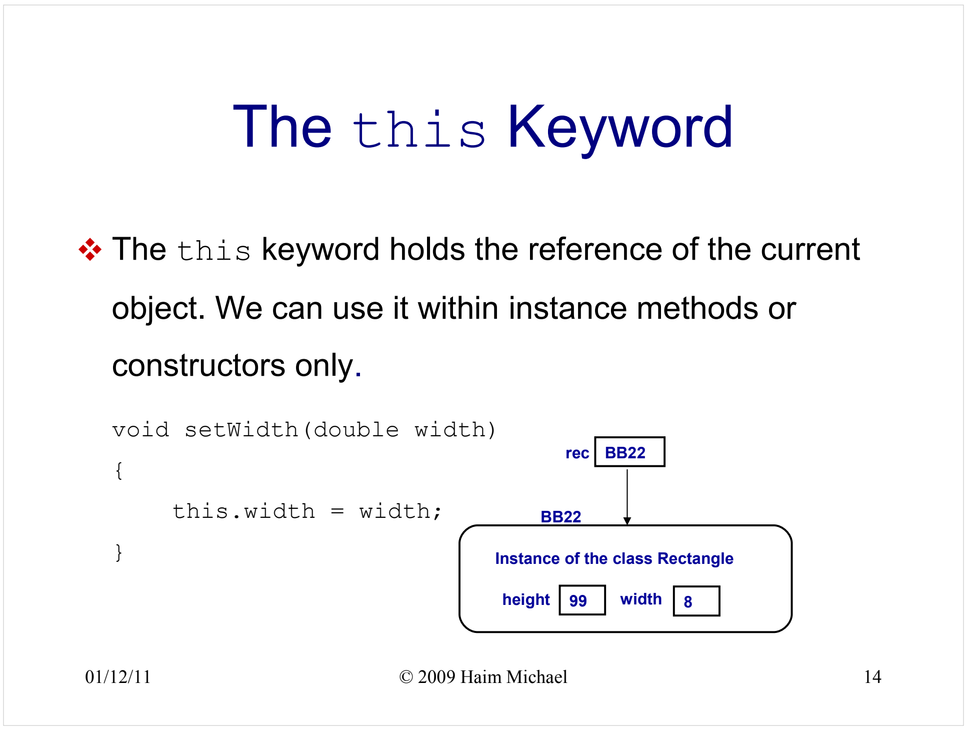


01/12/11 © 2009 Haim Michael 13

Given that the doSomething method was declared as follows:

```
class Xxx
{
            int var1, var2;
            void doSomething()
            {
                        System.out.println(var1+var2);
            }
}
```

The output will be: 20.



The `this` Keyword

❖ The `this` keyword holds the reference of the current object. We can use it within instance methods or constructors only.

```
void setWidth(double width)
{
     this.width = width;
}
```

rec
BB22
BB22
Instance of the class Rectangle
height
99
width
8

01/12/11 © 2009 Haim Michael 14

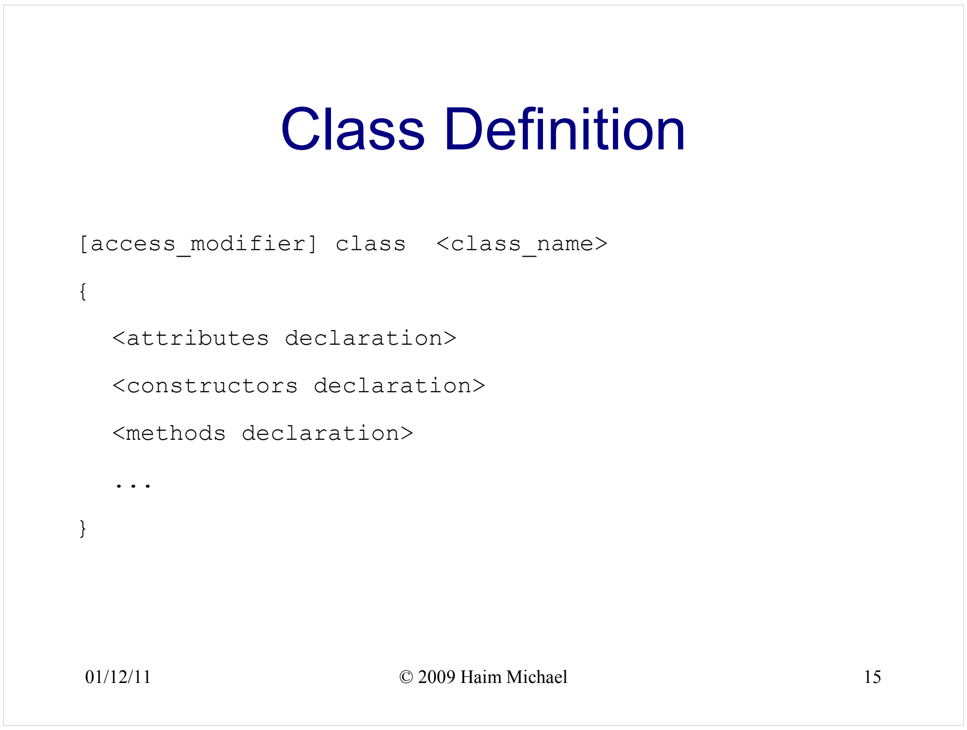
The option of placing the ‘this’ keyword within a constructor will be presented in the next slides.

Till now, the distinction between instance methods and static methods hasn’t been presented. All the methods that were discussed were instance methods.

The possible uses of the ‘this’ keyword are:

To reference local attribute and method members within a local method or constructor. This can be used in disambiguating a local method or constructor variable from variable which isn’t a local one.

To pass the reference of the current object as an argument to another method.



Class Definition

```
[access_modifier] class  <class_name>
{
   <attributes declaration>
   <constructors declaration>
   <methods declaration>
   ...
}
```

01/12/11 © 2009 Haim Michael 15

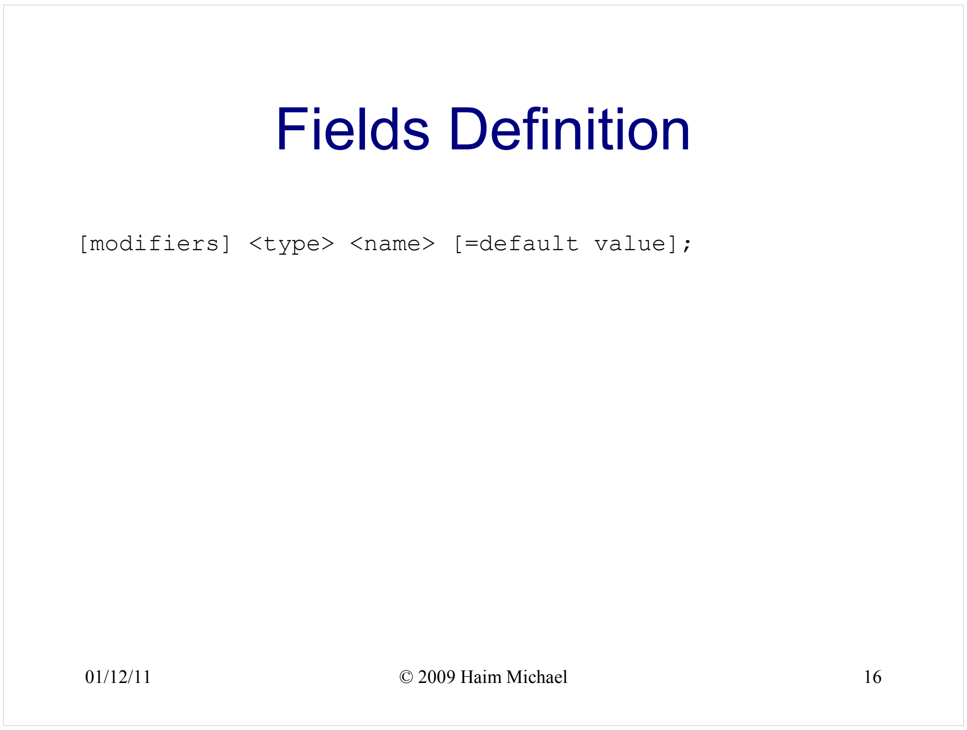
The access modifier (when declaring a normal class) might be public or package friendly. The package friendly access modifier is given automatically to class that its access modifier wasn’t mentioned. More explanations regarding the possible access modifiers will be given in the next slides.

The <class name> can be any legal name. The <class name> should start with an upper case.

The body of the class declares the set of attributes, constructors and methods that the class has.

Example:

```
public class Rectangle
{
                …
}
```



Fields Definition

```
[modifiers] <type> <name> [=default value];
```

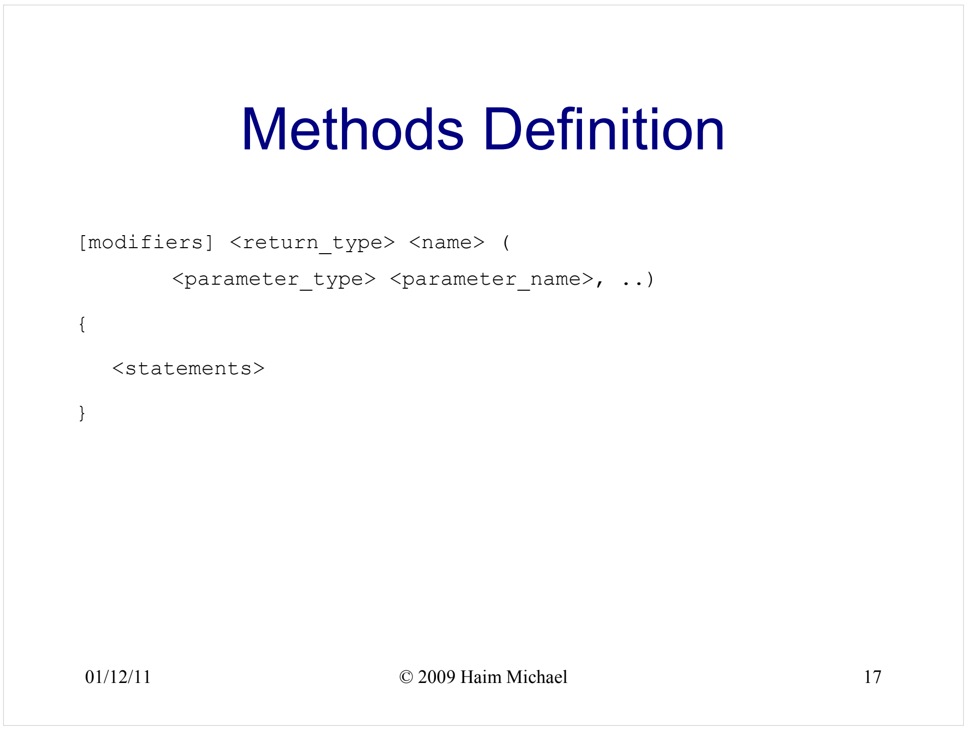
The <type> can be either a primitive type or a class type.

The <name> can be any legal identifier.

There are several possible [modifiers]. For the moment, use only the ‘public’ and the ‘private’ modifiers.

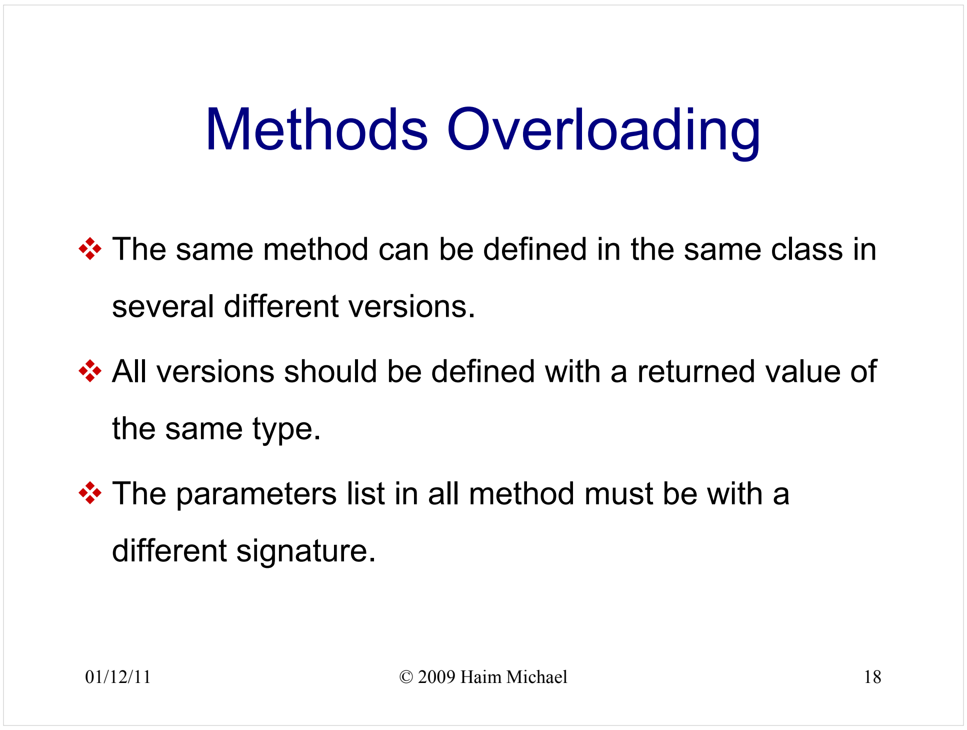
Example:

```
public int width=20;

public Point location;
```



Methods Definition

```
[modifiers] <return_type> <name> (
        <parameter_type> <parameter_name>, ..)
{
   <statements>
}
```



Methods Overloading

- ❖ The same method can be defined in the same class in several different versions.
- ❖ All versions should be defined with a returned value of the same type.
- ❖ The parameters list in all method must be with a different signature.

01/12/11 © 2009 Haim Michael 18

Java, the greatest language in the world, allows you to reuse a method name for more than one method. This works only if there is something in the method calls that distinguishes the method that is actually needed. The distinction is based on the number and/or type of the arguments.

By reusing the method name, you end up with several methods:

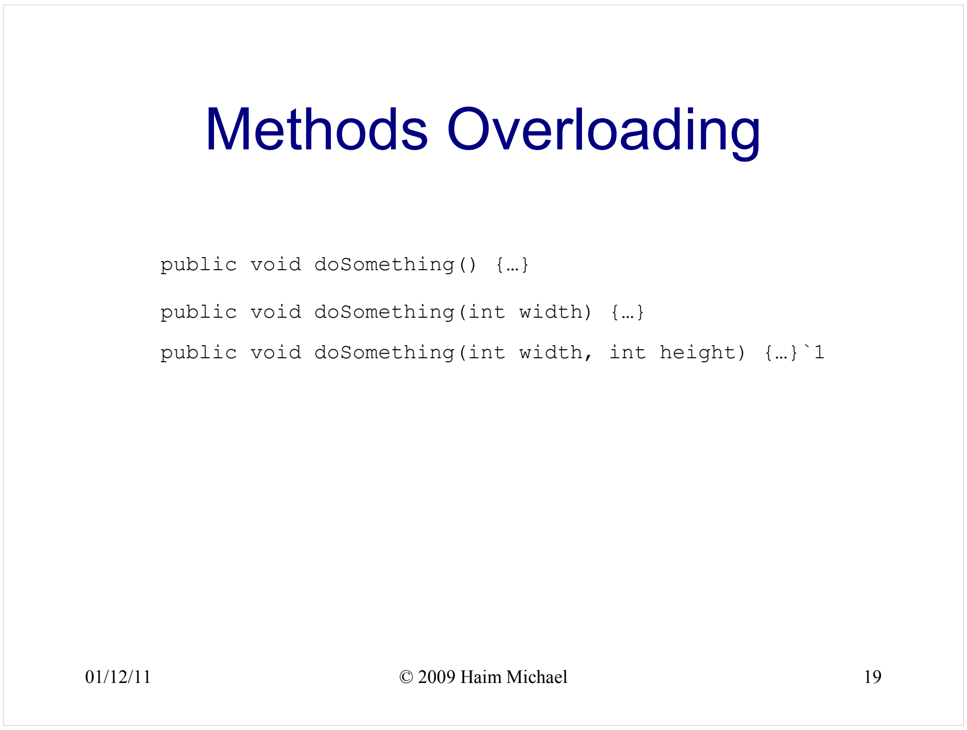
```
public void doSomething() {…  }
public void doSomething(int width) {…  }
public void doSomething(int width, int height) {… }
```

When you write code to call one of these methods, the appropriate method is chosen according to the type of arguments and/or the number of arguments that you send.

The rules that apply to overloaded methods are:

The argument lists of the calling statement must differ enough to allow unambiguous determination of the proper method to call.

The return type of the overloaded methods can be different, but it is not sufficient for the return type to be the only difference. The argument lists of overloaded methods must differ.



Methods Overloading

```
public void doSomething() {…}
public void doSomething(int width) {…}
public void doSomething(int width, int height) {…}`1
```

01/12/11 © 2009 Haim Michael 19

Java, the greatest language in the world, allows you to reuse a method name for more than one method. This works only if there is something in the method calls that distinguishes the method that is actually needed. The distinction is based on the number and/or type of the arguments.

By reusing the method name, you end up with several methods:

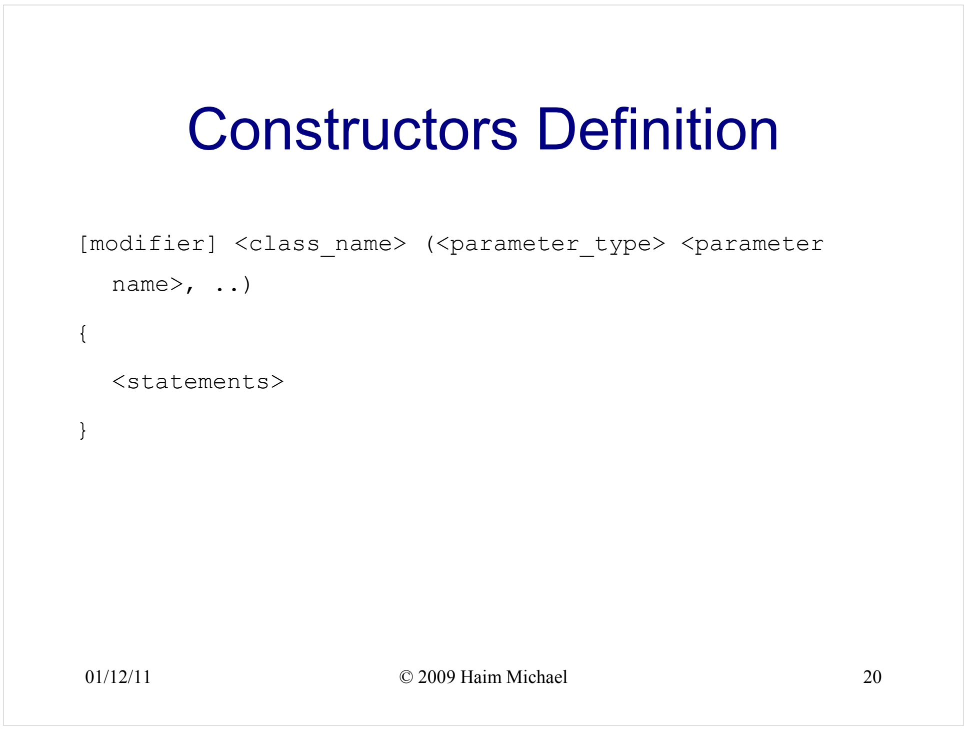
```
public void doSomething() {…  }
public void doSomething(int width) {…  }
public void doSomething(int width, int height) {… }
```

When you write code to call one of these methods, the appropriate method is chosen according to the type of arguments and/or the number of arguments that you send.

The rules that apply to overloaded methods are:

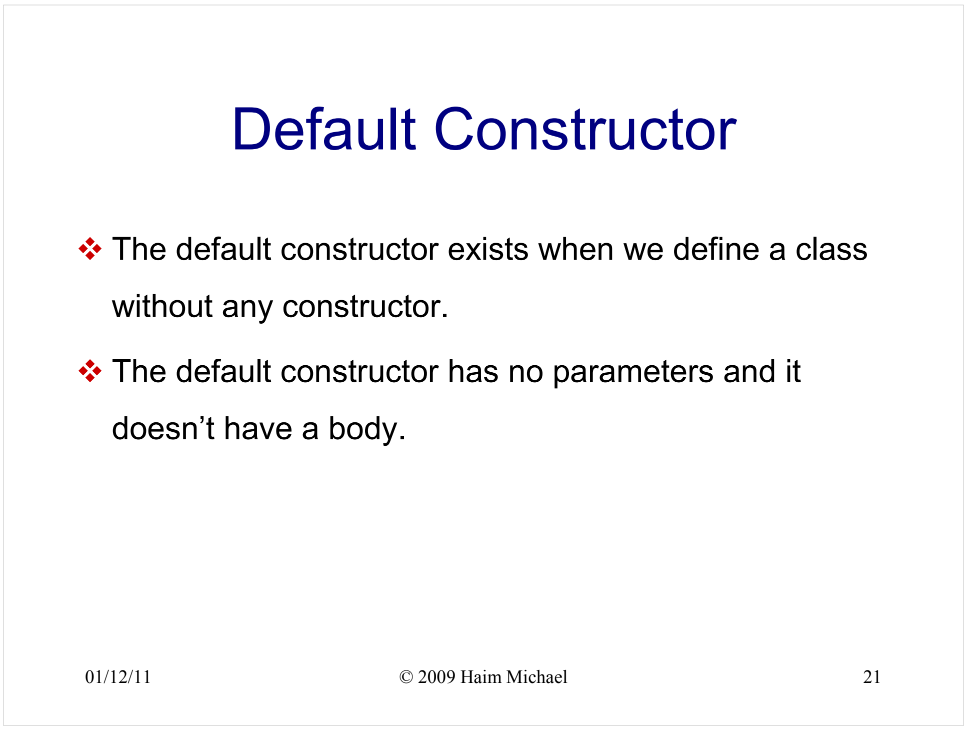
The argument lists of the calling statement must differ enough to allow unambiguous determination of the proper method to call.

The return type of the overloaded methods can be different, but it is not sufficient for the return type to be the only difference. The argument lists of overloaded methods must differ.



Constructors Definition

```
[modifier] <class_name> (<parameter_type> <parameter
  name>, ..)
{
  <statements>
}
```

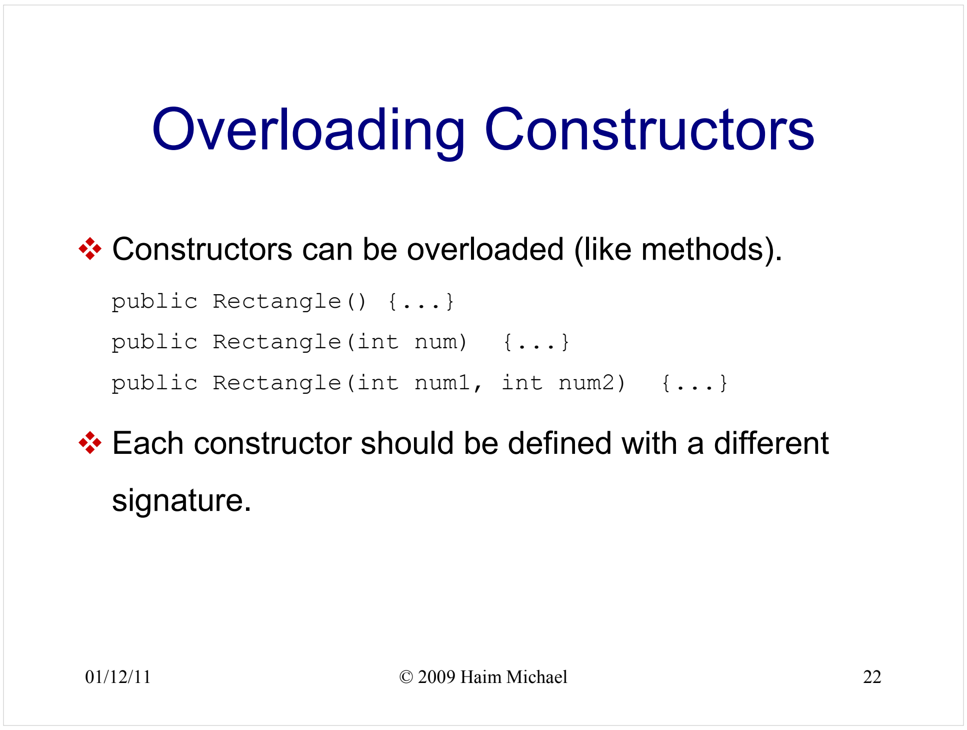


Default Constructor

- The default constructor exists when we define a class without any constructor.
- The default constructor has no parameters and it doesn't have a body.

When the user declares a constructor, the default constructor stops to exist.

Thanks to the default constructor it is possible instantiating a class without having to declare any constructor.



Overloading Constructors

- Constructors can be overloaded (like methods).

```
public Rectangle() {...}
public Rectangle(int num)  {...}
public Rectangle(int num1, int num2)  {...}
```

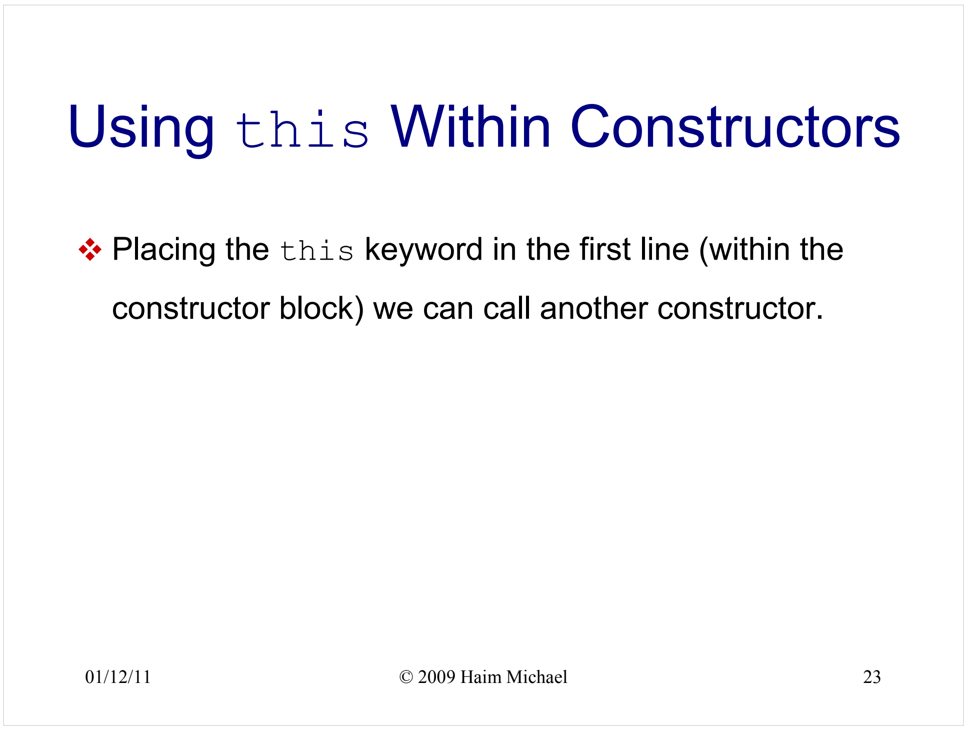
- Each constructor should be defined with a different signature.

01/12/11 © 2009 Haim Michael 22

Java, the greatest language in the world, allows you to declare more than one constructor in the same clas. This works only if there is something in the constructors calls that distinguishes the constructor that you actually want to invoke. The distinction is based on the number and type of the arguments.

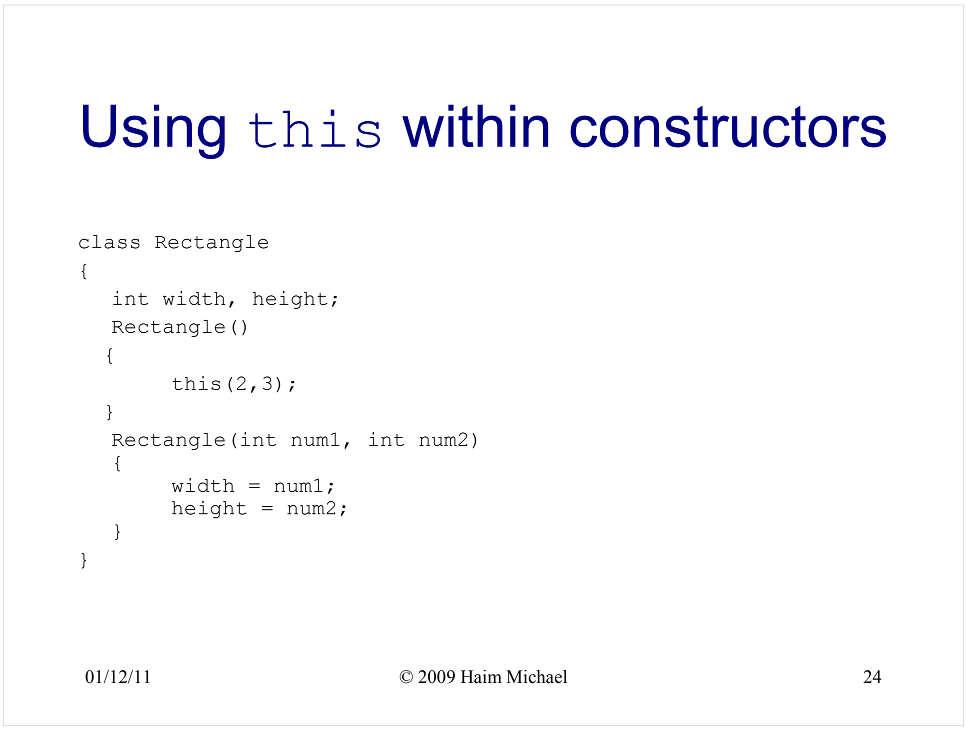
When you write code to call one of these constructors, the appropriate constructor is chosen according to the type of arguments or the number of arguments that you supply.

The argument lists of the calling statement must differ enough to allow unambiguous determination of the proper constructor to call.



Using `this` Within Constructors

- Placing the `this` keyword in the first line (within the constructor block) we can call another constructor.

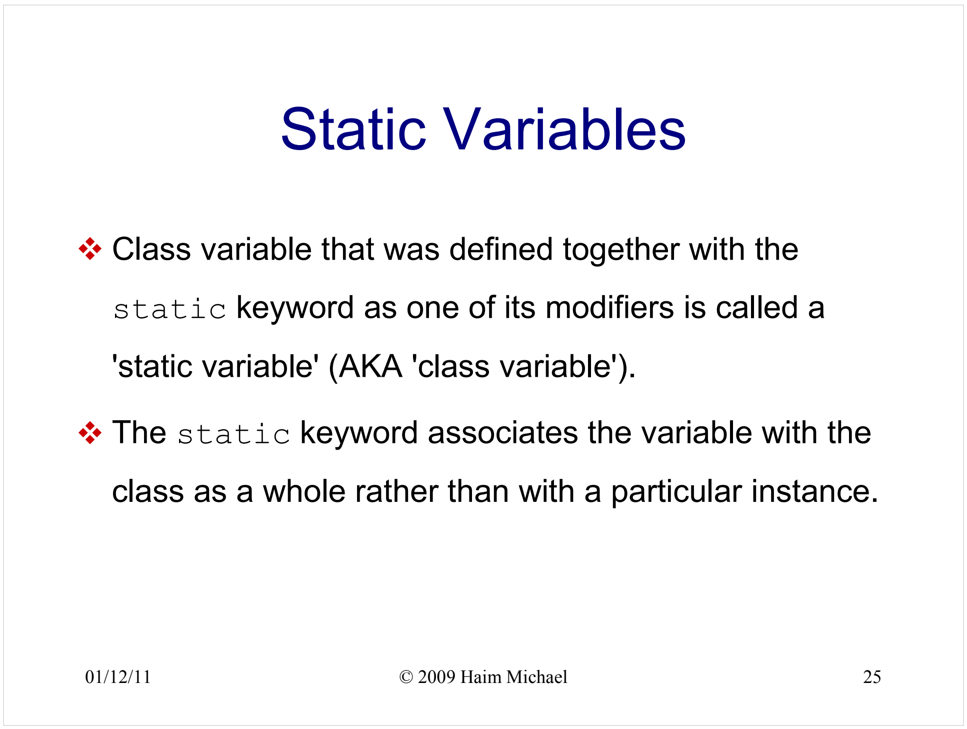


Using `this` within constructors

```
class Rectangle
{
   int width, height;
   Rectangle()
  {
        this(2,3);
  }
   Rectangle(int num1, int num2)
   {
        width = num1;
        height = num2;
   }
}
```

01/12/11 © 2009 Haim Michael 24

Using this technique it is possible declaring a class with several constructors where as only the constructor with the largest number of parameters has a concrete body. All the others will call him using ‘this’.



Static Variables

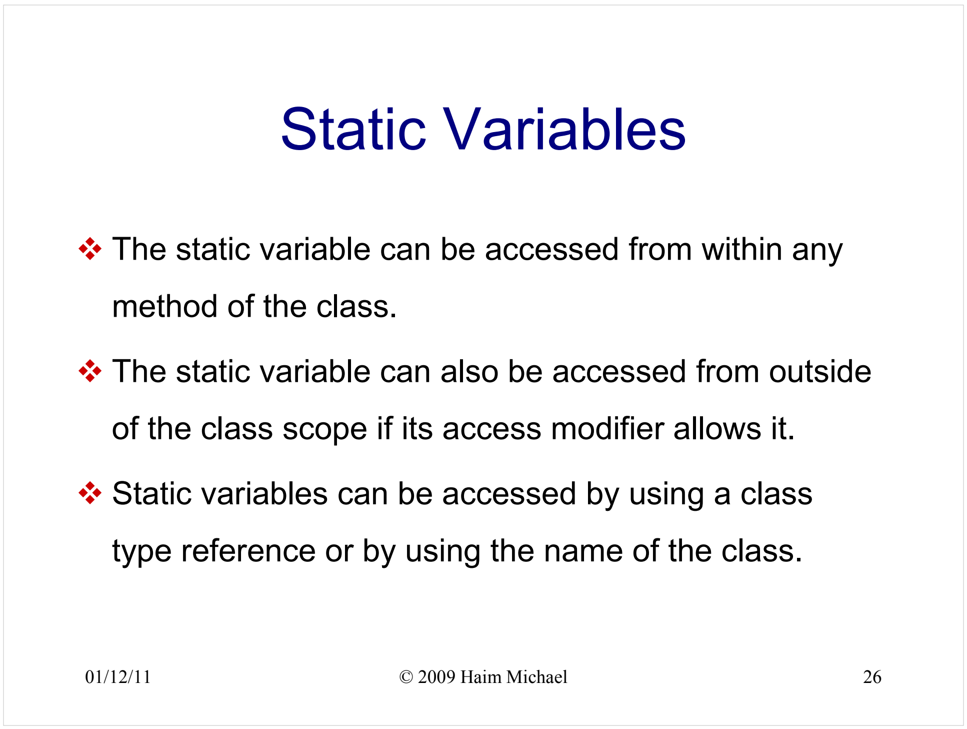
- Class variable that was defined together with the `static` keyword as one of its modifiers is called a 'static variable' (AKA 'class variable').
- The `static` keyword associates the variable with the class as a whole rather than with a particular instance.

01/12/11 © 2009 Haim Michael 25

A class attribute, that isn’t a static variable, is called an “instance variable”. After all, it is associated with a specific instance.

When it is desirable to have a variable that is shared among all instances of a class the variable will be declared as a static variable. This can be used as a mean for communication between instances or to keep track of the number of instances that have been created.

A static variable is similar in some ways to a global variable in other languages. Java doesn’t have global variables. The static variable is kind of a replacement to global variable.



Static Variables

- The static variable can be accessed from within any method of the class.
- The static variable can also be accessed from outside of the class scope if its access modifier allows it.
- Static variables can be accessed by using a class type reference or by using the name of the class.

The following example presents the possibilities of accessing a static variable:

```
Class Rectangle
{
            String name;
            static int numOfRectangles=0;
            Rectangle()
            {
                        this("NoName");
            }
            Rectangle(String str)
            {
                        numOfRectangles++;
                        name = str;
            }
}
```

```
public static void main(String args[])
{
            Rectangle myRectangle, hisRectangle;
            myRectangle = new Rectangle();
            hisRectangle = new Rectangle("Juka");
            System.out.println(
              "The number of Rectangles instances is : "
              + Rectangle.numOfRectangles);
}
```

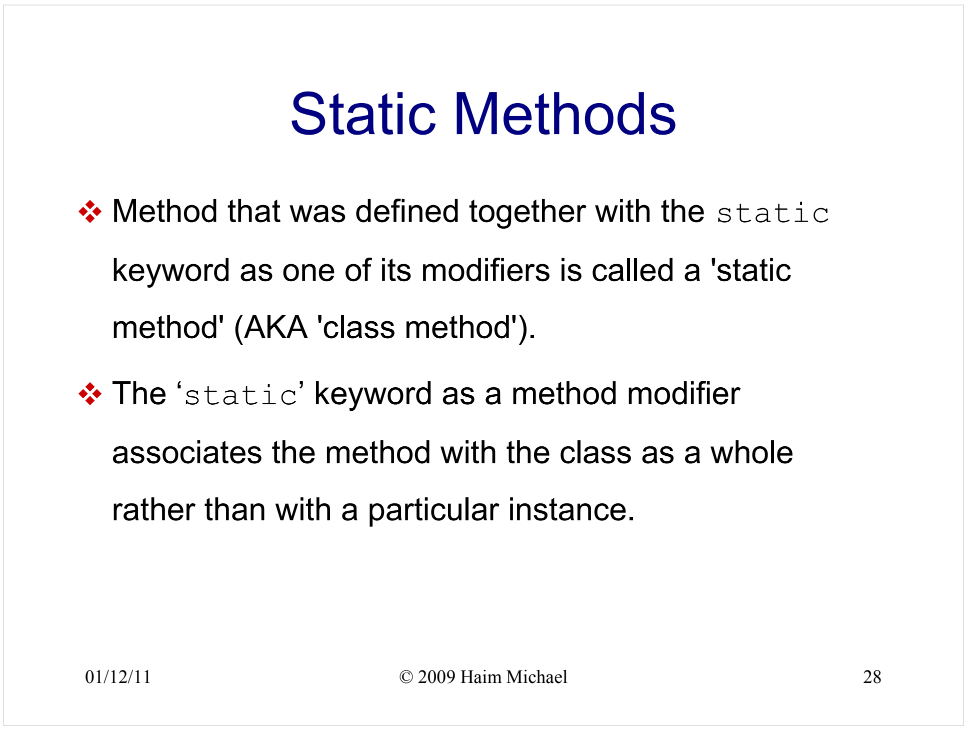
The System.out.println statement could also look as:

System.out.println("The number of Xxx instances is : " + numOfRectangle);

Because it is possible to call a static method without having any instance of the class to which it belongs, it isn't possible placing the 'this' keyword within the static method. Further more, attempting to access non-static variables causes a compiler error.

Non static variables (means: instance variables) are bound to specific instance and can be accessed only through instance references.

Notice that you can't override a static method. (The overriding mechanism will be described in the Inheritance module). The 'main' method is a static method because the java.exe utility program doesn't create an instance from the class that it treats as a stand alone application.



Static Methods

- Method that was defined together with the `static` keyword as one of its modifiers is called a 'static method' (AKA 'class method').
- The '`static`' keyword as a method modifier associates the method with the class as a whole rather than with a particular instance.

01/12/11 © 2009 Haim Michael 28

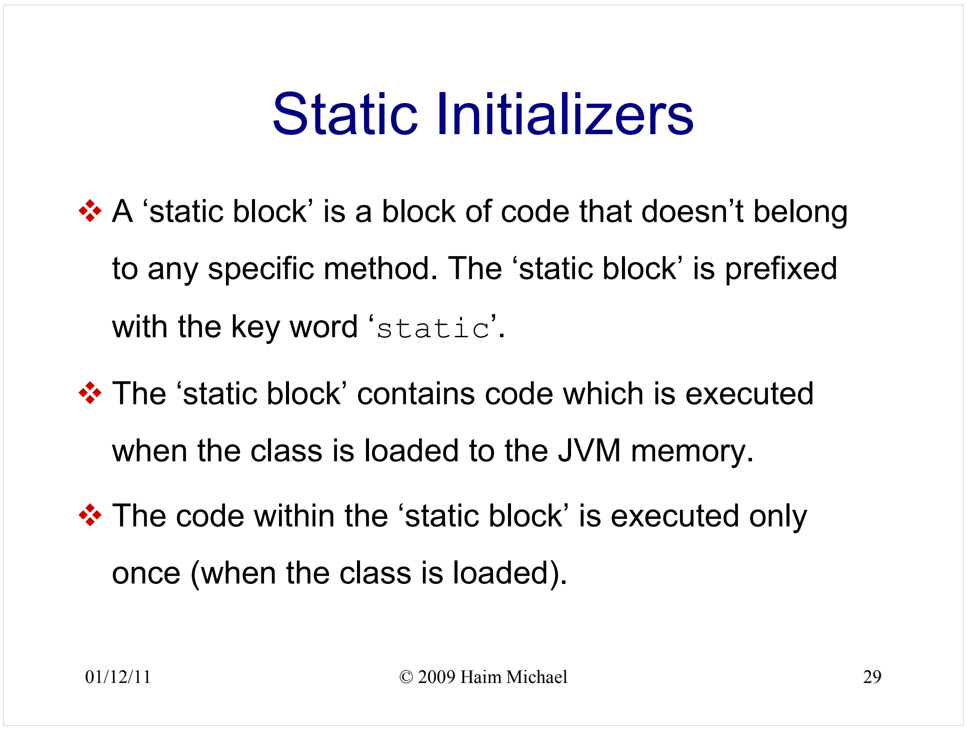
One of the well known static methods is the "main" method which you declare in a class in order to use that class as application.

A static method can be called without instantiating the class. As with static variables, you can access static methods in two ways:

1. Placing the name of the class before the method (with a separated dot(.)).
2. Placing a class type reference before the method (with a separated dot(.)).

```
public class Rectangle
{
                static int numOfRectangle;
                …
                public static int getNumOfRectangles()
                {
                                return numOfRectangles;
                }
}
```

```
…
System.out.println("num of Rectangle instances is : " +
Rectangle.getNumOfRectangles());
…
System.out.println("num of Rectangle instances is : " +
myRectangle.getNumOfXxx());
```



Static Initializers

- A ‘static block’ is a block of code that doesn’t belong to any specific method. The ‘static block’ is prefixed with the key word ‘`static`’.
- The ‘static block’ contains code which is executed when the class is loaded to the JVM memory.
- The code within the ‘static block’ is executed only once (when the class is loaded).

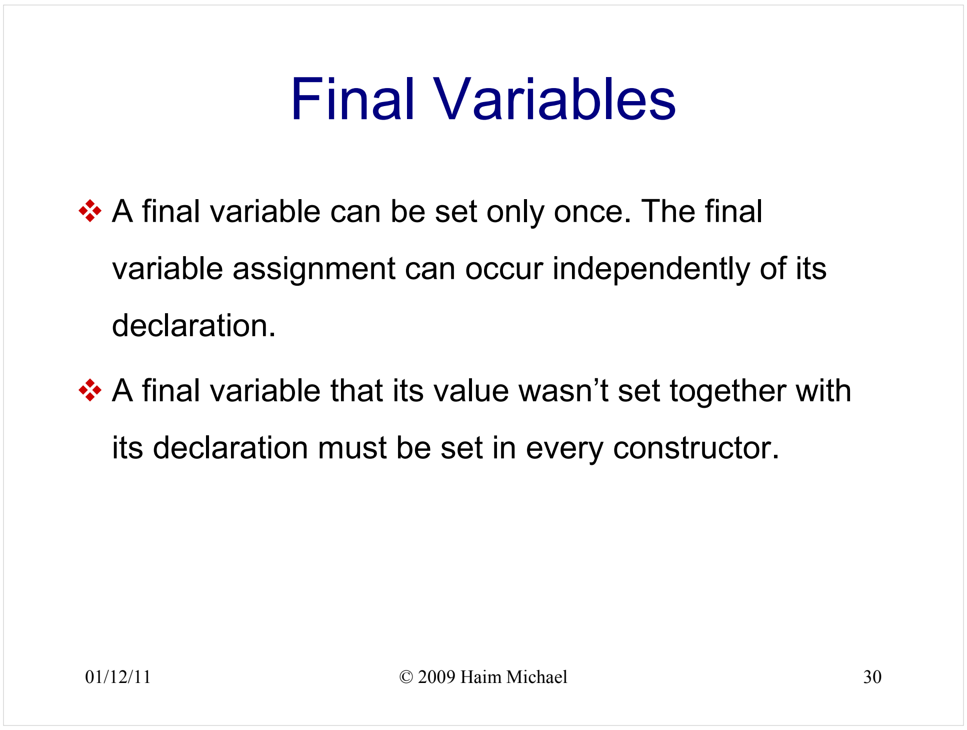
01/12/11 © 2009 Haim Michael 29

The static block is usually used to initialize static (class) attributes.

If the class has several ‘static blocks’ then they are executed in the order of their appearance in the class.

Example:

```
class Rectangle
{
        static int numOfRectangles;
        static int defaultRectangleId;
        static
        {
                numOfRectangles = 0;
                defaultRectangleId = 123123;
        }
        …
}
```

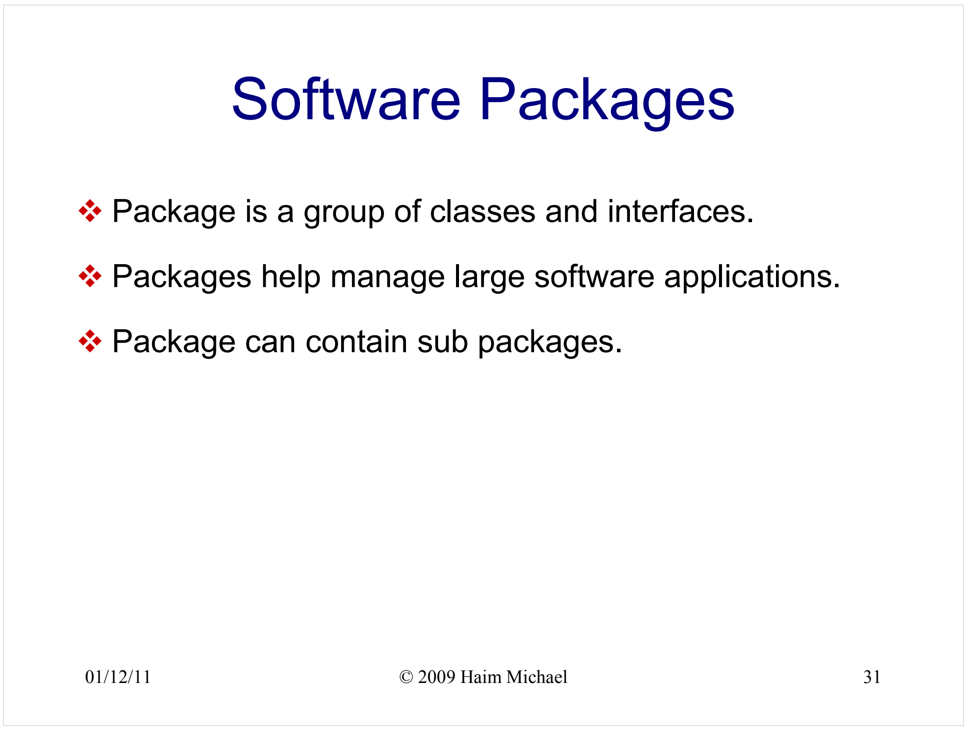


Final Variables

- A final variable can be set only once. The final variable assignment can occur independently of its declaration.
- A final variable that its value wasn't set together with its declaration must be set in every constructor.

A variable that was marked as 'final' becomes constant. Any attempt of changing the value of a 'final' variable causes a compiler error.

However, you should notice that a class type variable that was marked as 'final' can't change the reference it holds. The object which this is its reference can be freely changed. Only the reference itself is final.



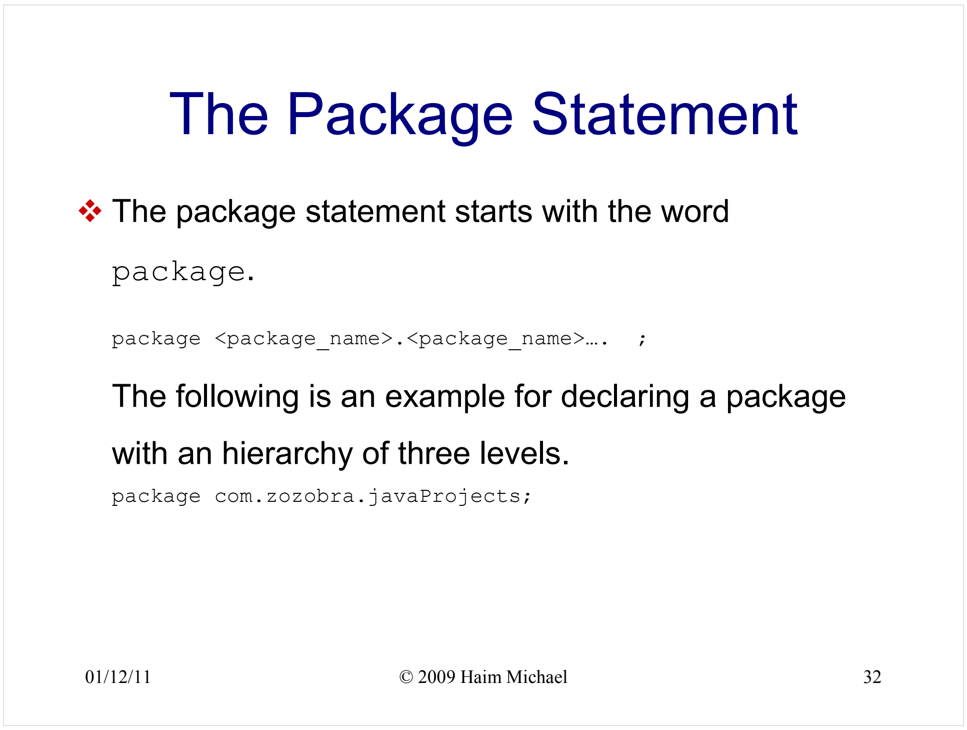
Software Packages

- Package is a group of classes and interfaces.
- Packages help manage large software applications.
- Package can contain sub packages.

The common technique is to group classes into a package by semantic similarity.

It is common to create hierarchy of packages that starts with package and sub package that their names is the company domain name.

If, for instance, a company named Zozobra that her web site URL is: www.zozobra.com wants to develop java classes then her hierarchy of packages will start with: com.zozobra.



The Package Statement

- The package statement starts with the word `package`.

  ```
  package <package_name>.<package_name>….  ;
  ```

  The following is an example for declaring a package with an hierarchy of three levels.

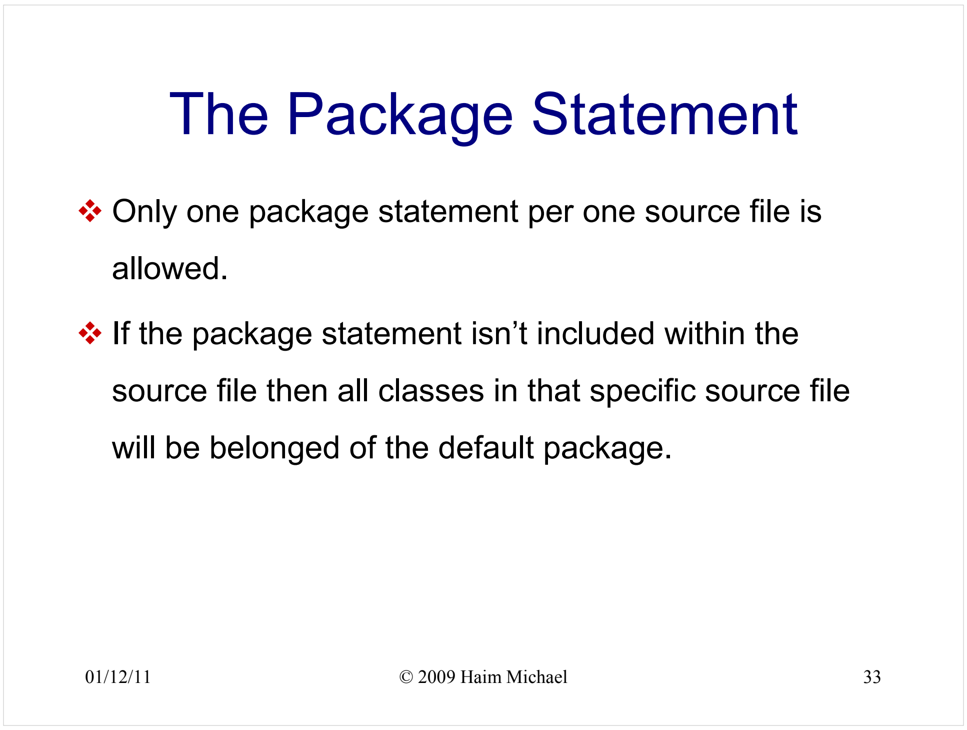
  ```
  package com.zozobra.javaProjects;
  ```

01/12/11 © 2009 Haim Michael 32

The common technique is to group classes into a package by semantic similarity.

It is common to create hierarchy of packages that starts with package and sub package that their names is the company domain name.

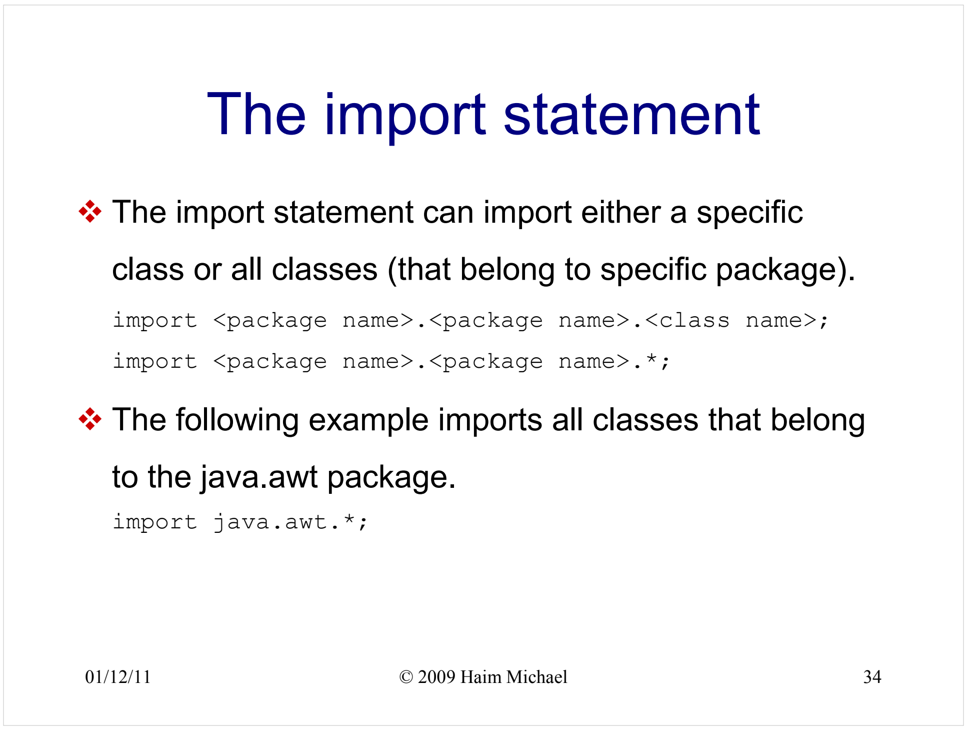
If, for instance, a company named Zozobra that her web site URL is: www.zozobra.com wants to develop java classes then her hierarchy of packages will start with: com.zozobra.



The Package Statement

- Only one package statement per one source file is allowed.
- If the package statement isn’t included within the source file then all classes in that specific source file will be belonged of the default package.

01/12/11 © 2009 Haim Michael 33



The import statement

- The import statement can import either a specific class or all classes (that belong to specific package).

```
import <package name>.<package name>.<class name>;
import <package name>.<package name>.*;
```

- The following example imports all classes that belong to the java.awt package.

```
import java.awt.*;
```

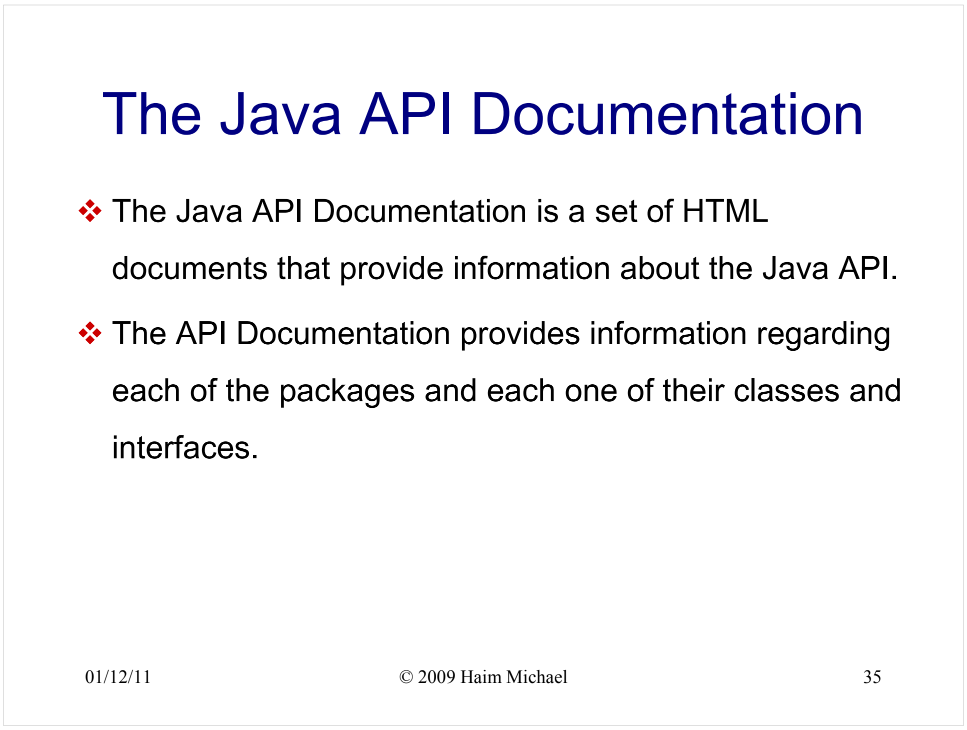
01/12/11 © 2009 Haim Michael 34

The import statement tells the compiler where to find the classes.

If the import statement isn't placed in the source code then all of classes that the import statement referred to should be written in their full qualified name. For instance, if the import statement: import java.awt.*; isn't placed in the source file then all of the classes that belong to the java.awt package should be written with the package name preceding their name. Therefore, the class Button will have to be written as java.awt.Button.

The import statement must be written after the package statement and before the class declarations.

It is a common mistake to consider the import statement as the include statement in C\C++. The using of the import statement with "*" doesn't effect performance. The use of import statements doesn't load the code as the use of the include statement in C\C++.

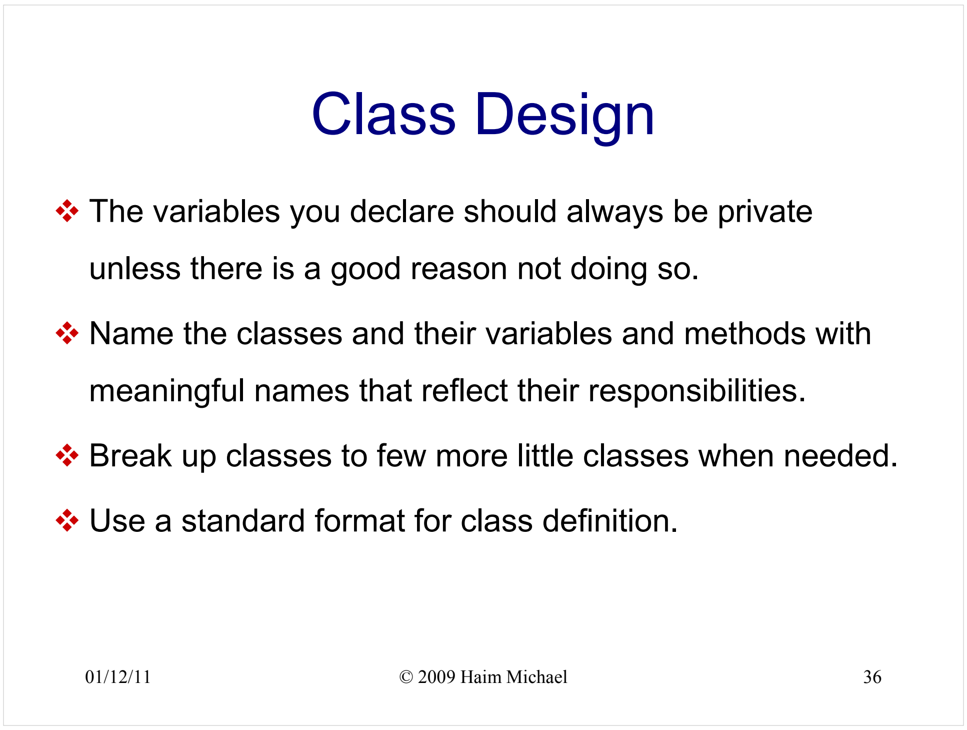


The Java API Documentation

- The Java API Documentation is a set of HTML documents that provide information about the Java API.
- The API Documentation provides information regarding each of the packages and each one of their classes and interfaces.

01/12/11 © 2009 Haim Michael 35

The API Documentation can be downloaded from http://java.sun.com and can be also browsed on-line.



Class Design

- ❖ The variables you declare should always be private unless there is a good reason not doing so.
- ❖ Name the classes and their variables and methods with meaningful names that reflect their responsibilities.
- ❖ Break up classes to few more little classes when needed.
- ❖ Use a standard format for class definition.

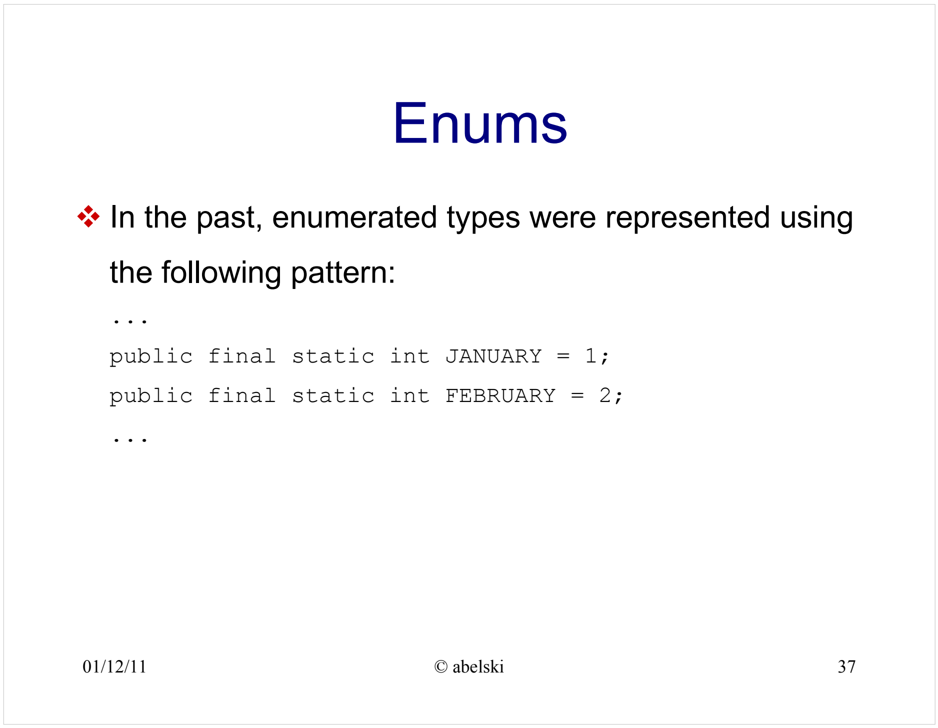
01/12/11 © 2009 Haim Michael 36

When declaring variables within a class, set their access modifier to be private and define the required methods with which these variables will be accessed and set. By doing so, you can make sure that these variables hold legal values.

Giving names that reflect the responsibilities (of the class, variable, method or package) makes the code more clear.

Be aware and avoid declaring classes which are too big. In these cases, you can always consider breaking the big class into smaller classes. That helps maintaining the code.

Using a standard format declaring classes improves the code clarity.



Enums

- In the past, enumerated types were represented using the following pattern:

```
...
public final static int JANUARY = 1;
public final static int FEBRUARY = 2;
...
```

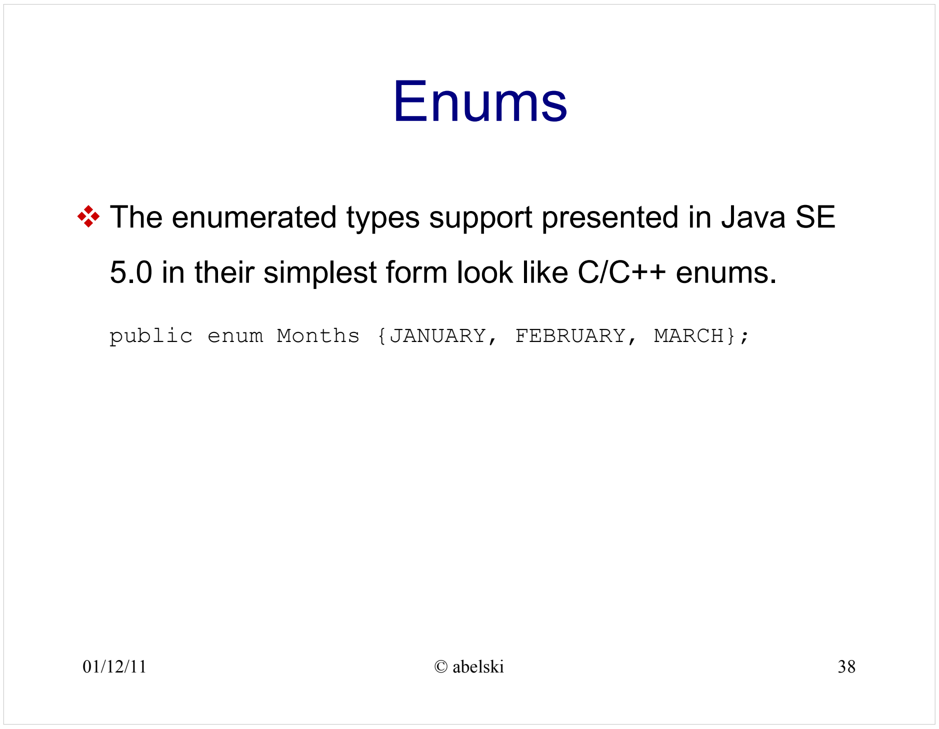
Using the Enum pattern has the following problems:

1. The programmer can avoid using the final variables and pass in a simple int value, which can be a different value from the expected one (overtime changes in the values used as the final static variables can be problematic).

2. Using the Enum pattern might cause to collisions with other int enum types that use the same name. As a result, there is a need to unique the chosen names (one option is adding a meaningful prefix).

3. The final variables used as enums are compile time constants. Overtime changes in the values used for the enums will require us re compile all source codes that use these enums.

4. Printing the value of a final int variable we use as enum is not informatice.



Enums

- The enumerated types support presented in Java SE 5.0 in their simplest form look like C/C++ enums.

```
public enum Months {JANUARY, FEBRUARY, MARCH};
```

01/12/11 © abelski 38

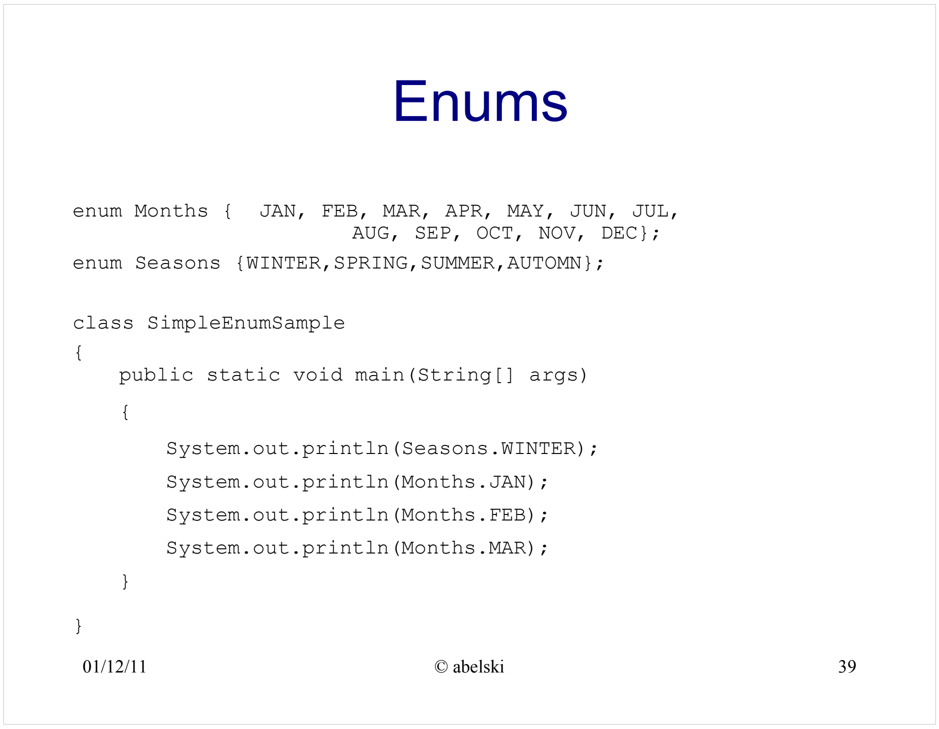
Using the Enum pattern has the following problems:

1. The programmer can avoid using the final variables and pass in a simple int value, which can be a different value from the expected one (overtime changes in the values used as the final static variables can be problematic).

2. Using the Enum pattern might cause to collisions with other int enum types that use the same name. As a result, there is a need to unique the chosen names (one option is adding a meaningful prefix).

3. The final variables used as enums are compile time constants. Overtime changes in the values used for the enums will require us re compile all source codes that use these enums.

4. Printing the value of a final int variable we use as enum is not informatice.

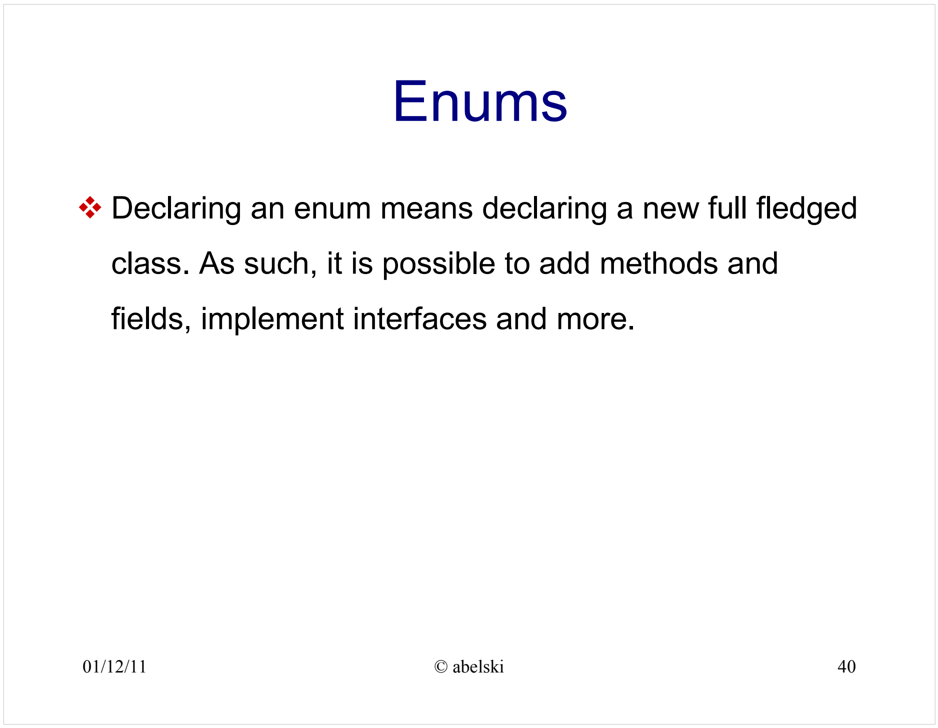


Enums

```
enum Months {  JAN, FEB, MAR, APR, MAY, JUN, JUL,
                      AUG, SEP, OCT, NOV, DEC};
enum Seasons {WINTER,SPRING,SUMMER,AUTOMN};

class SimpleEnumSample
{
    public static void main(String[] args)
    {
        System.out.println(Seasons.WINTER);
        System.out.println(Months.JAN);
        System.out.println(Months.FEB);
        System.out.println(Months.MAR);
    }
}
```

01/12/11 © abelski 39

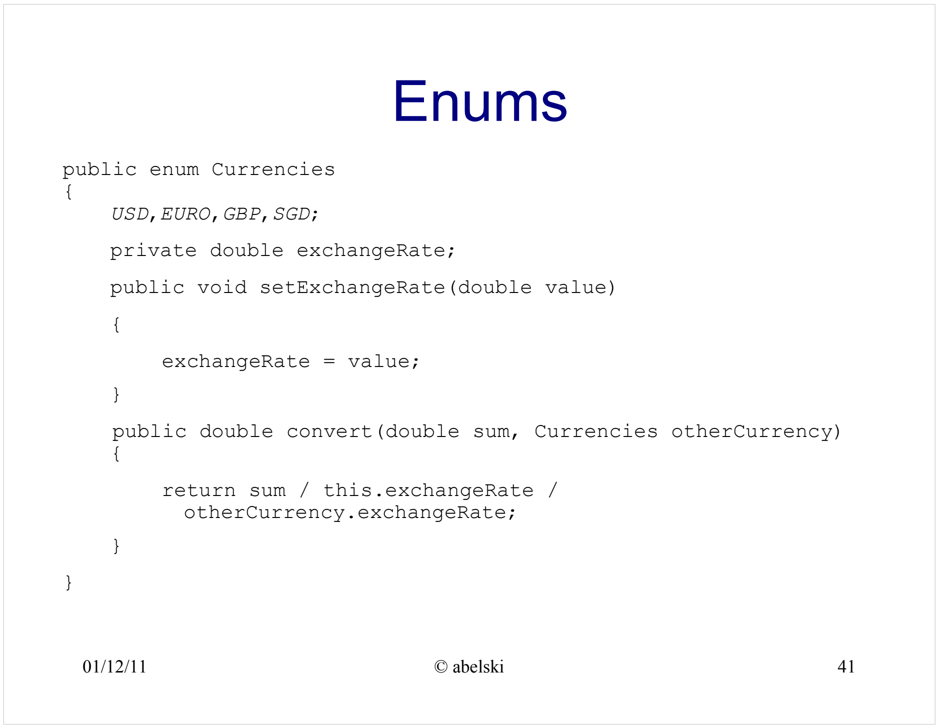
You can find the above code sample ready for download in the samples folder of this topic.



Enums

❖ Declaring an enum means declaring a new full fledged class. As such, it is possible to add methods and fields, implement interfaces and more.

01/12/11 © abelski 40

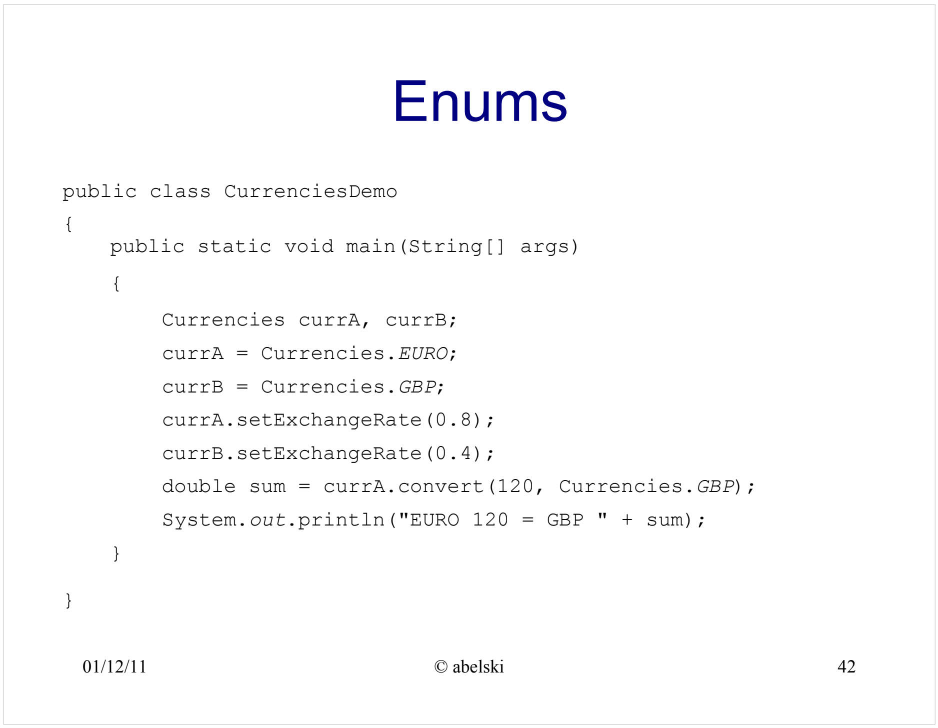


Enums

```
public enum Currencies
{
    USD,EURO,GBP,SGD;

    private double exchangeRate;

    public void setExchangeRate(double value)

    {

        exchangeRate = value;

    }

    public double convert(double sum, Currencies otherCurrency)
    {

        return sum / this.exchangeRate /
          otherCurrency.exchangeRate;

    }

}
```

01/12/11 © abelski 41

You can download the source code of the currencies sample from the samples folder of this topic.



Enums

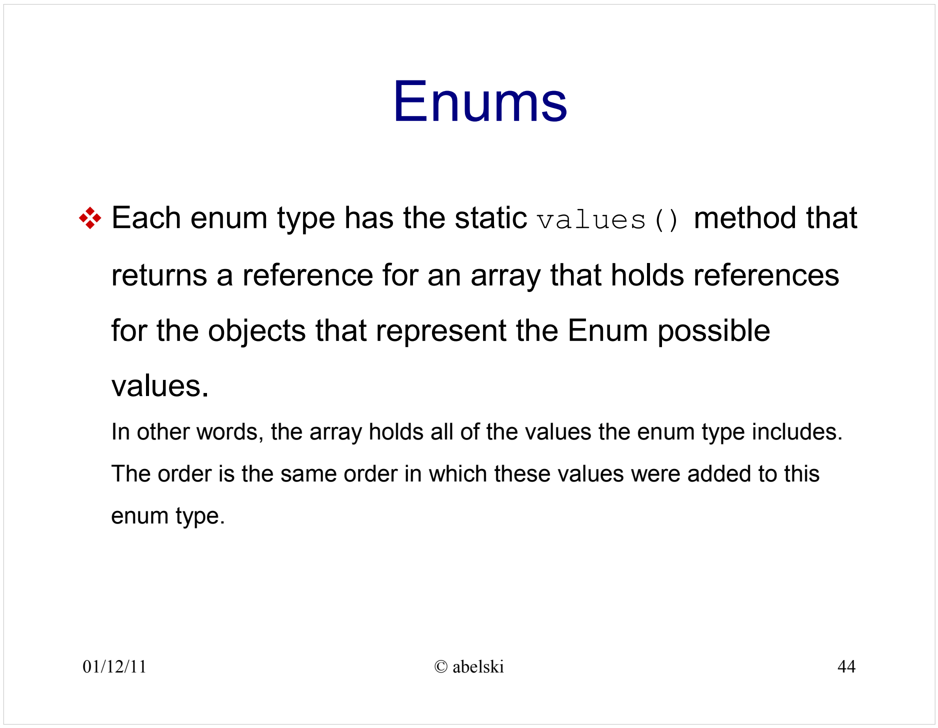
```
public class CurrenciesDemo
{
    public static void main(String[] args)
    {
        Currencies currA, currB;
        currA = Currencies.EURO;
        currB = Currencies.GBP;
        currA.setExchangeRate(0.8);
        currB.setExchangeRate(0.4);
        double sum = currA.convert(120, Currencies.GBP);
        System.out.println("EURO 120 = GBP " + sum);
    }
}
```

01/12/11 © abelski 42

You can download the source code of the currencies sample from the samples folder of this topic.

Enums

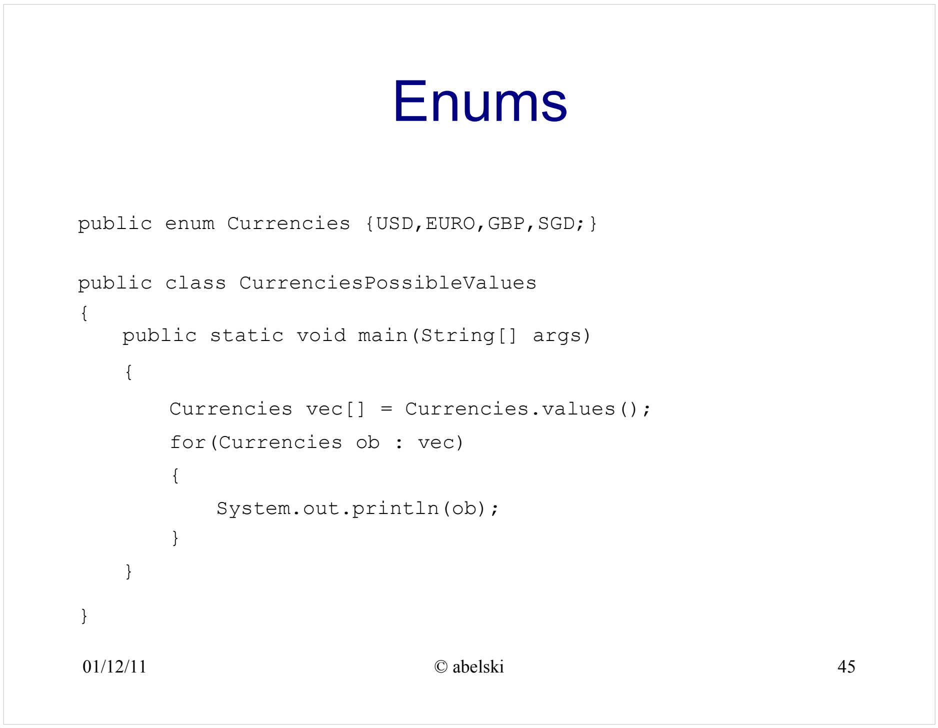
- The enum types provide high-quality implementations of all the Object methods (e.g. `toString()`).
- All enum types are both `Comparable` and `Serializable`.



Enums

- Each enum type has the static `values()` method that returns a reference for an array that holds references for the objects that represent the Enum possible values.

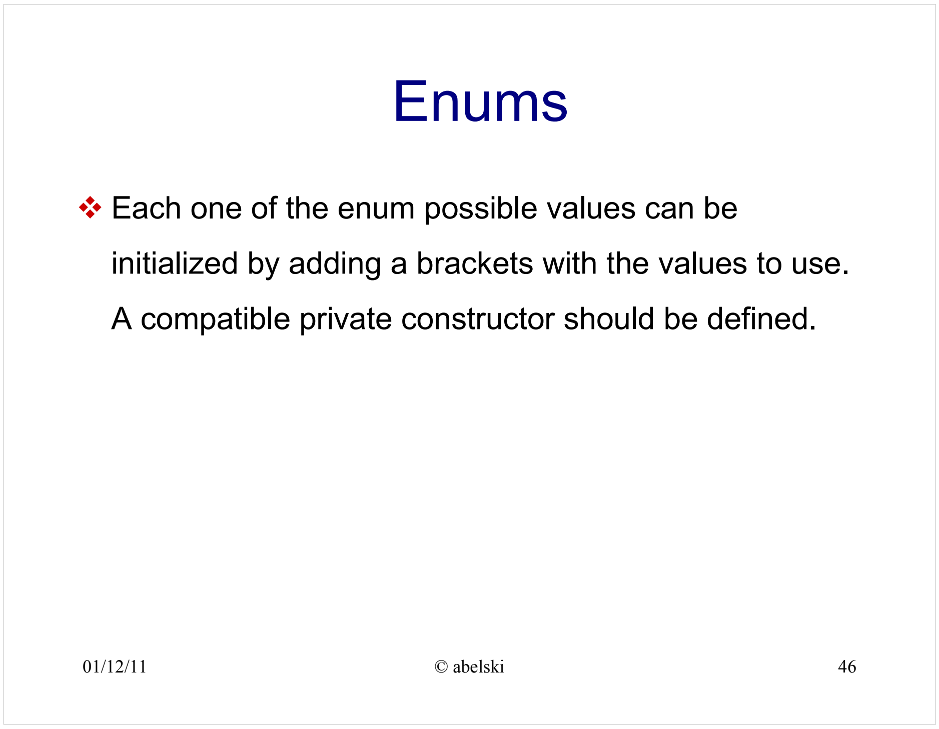
  In other words, the array holds all of the values the enum type includes. The order is the same order in which these values were added to this enum type.



Enums

```
public enum Currencies {USD,EURO,GBP,SGD;}

public class CurrenciesPossibleValues
{
    public static void main(String[] args)
    {
        Currencies vec[] = Currencies.values();
        for(Currencies ob : vec)
        {
            System.out.println(ob);
        }
    }
}
```

You can download the source code of this sample from the samples folder of this topic.

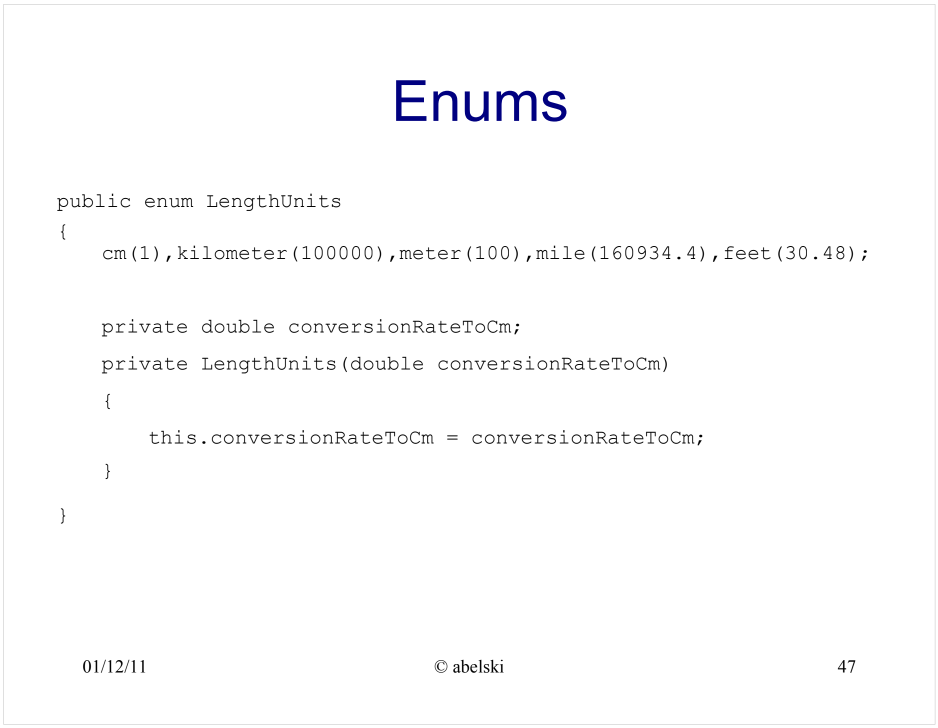


Enums

- ❖ Each one of the enum possible values can be initialized by adding a brackets with the values to use. A compatible private constructor should be defined.

01/12/11 © abelski 46

When declaring a new enum type with a constructor, the constructor must be private. It can't have any other access level. It can be private only.

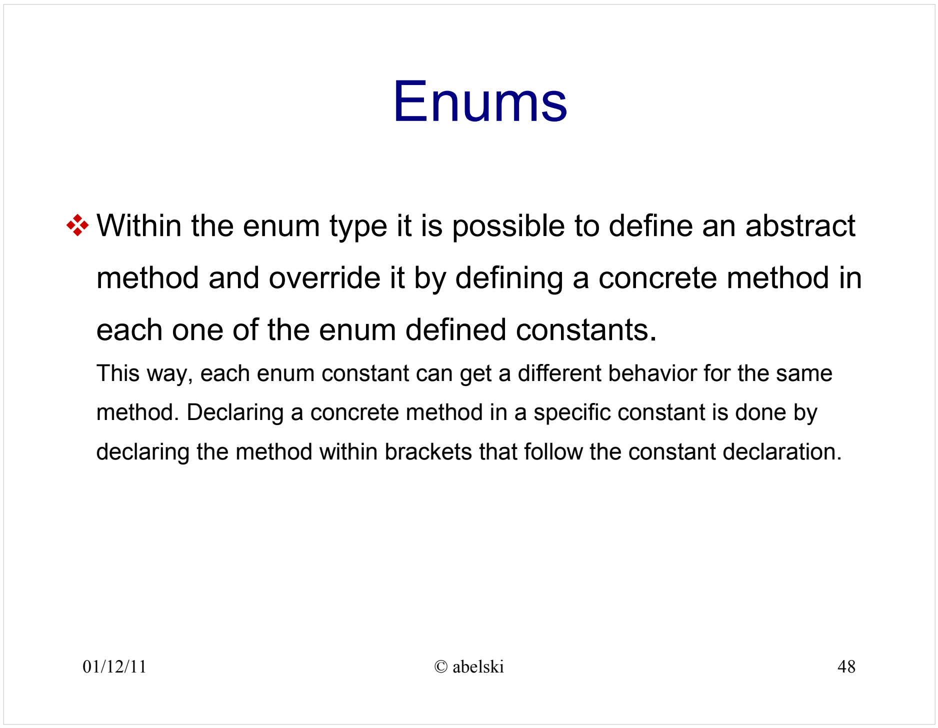


Enums

```
public enum LengthUnits
{
    cm(1),kilometer(100000),meter(100),mile(160934.4),feet(30.48);

    private double conversionRateToCm;
    private LengthUnits(double conversionRateToCm)
    {
        this.conversionRateToCm = conversionRateToCm;
    }
}
```

01/12/11 © abelski 47

You can find the LengthUnits.java sample inside the samples folder of this topic.



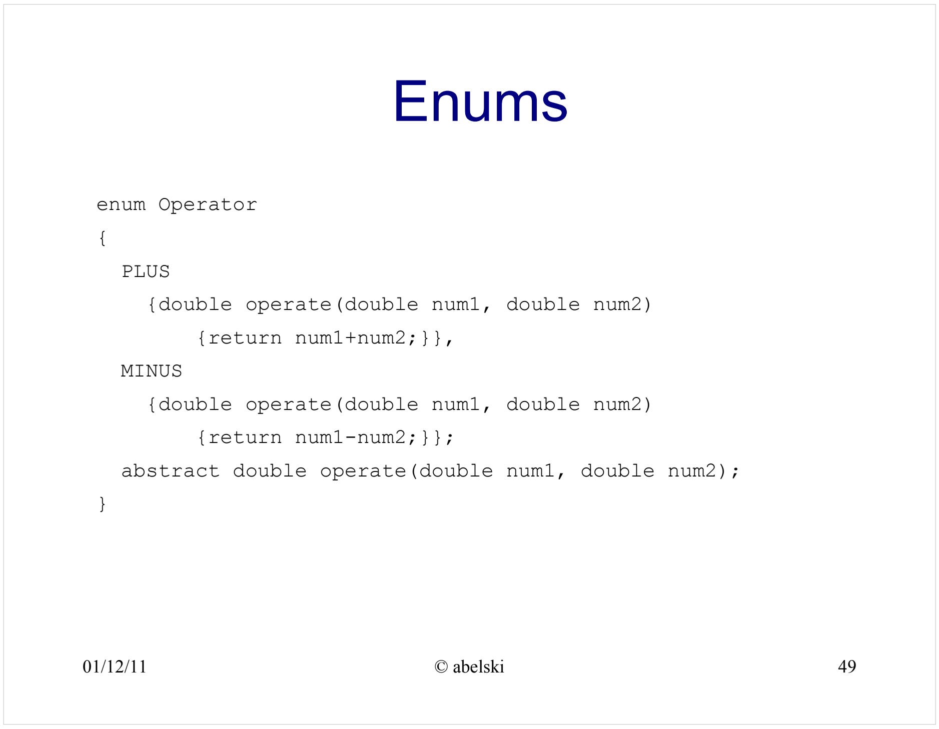
Enums

❖ Within the enum type it is possible to define an abstract method and override it by defining a concrete method in each one of the enum defined constants.
This way, each enum constant can get a different behavior for the same method. Declaring a concrete method in a specific constant is done by declaring the method within brackets that follow the constant declaration.

01/12/11 © abelski 48

The common name for such methods is “constant specific
methods”.

You can download the complete source code files of the above
sample from the samples folder of this topic. You will find the
files within the Mathematc_Operators_Demo sub folder.



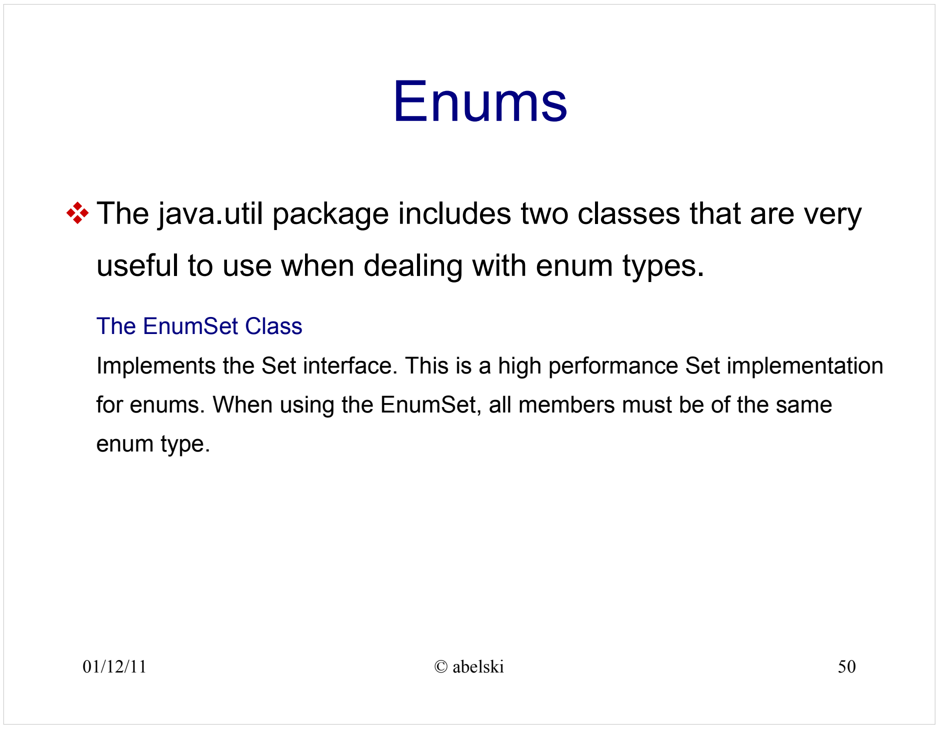
Enums

```
enum Operator
{
  PLUS
    {double operate(double num1, double num2)
        {return num1+num2;}},
  MINUS
    {double operate(double num1, double num2)
        {return num1-num2;}};
  abstract double operate(double num1, double num2);
}
```

01/12/11 © abelski 49

The common name for such methods is “constant specific
methods”.

You can download the complete source code files of the above
sample from the samples folder of this topic. You will find the
files within the Mathematc_Operators_Demo sub folder.



Enums

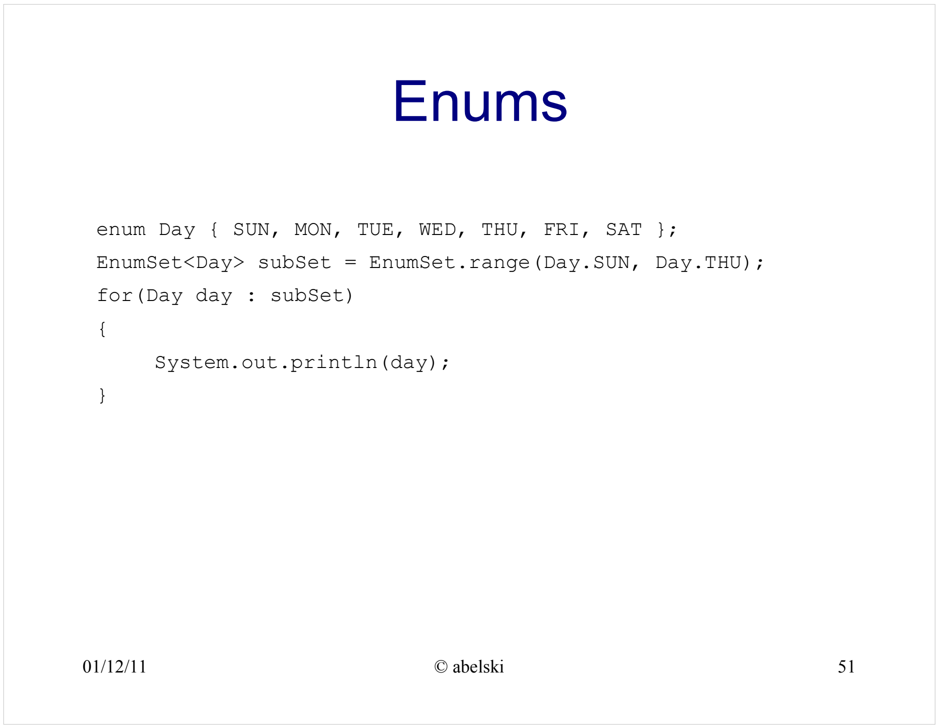
❖ The java.util package includes two classes that are very useful to use when dealing with enum types.

The EnumSet Class
Implements the Set interface. This is a high performance Set implementation for enums. When using the EnumSet, all members must be of the same enum type.

01/12/11 © abelski 50

Internally, the EnumSet object uses a bit vector. Using this
simple bit vector enables its high performance.

The complete source code of the above sample can be found
within the samples folder of this topic.



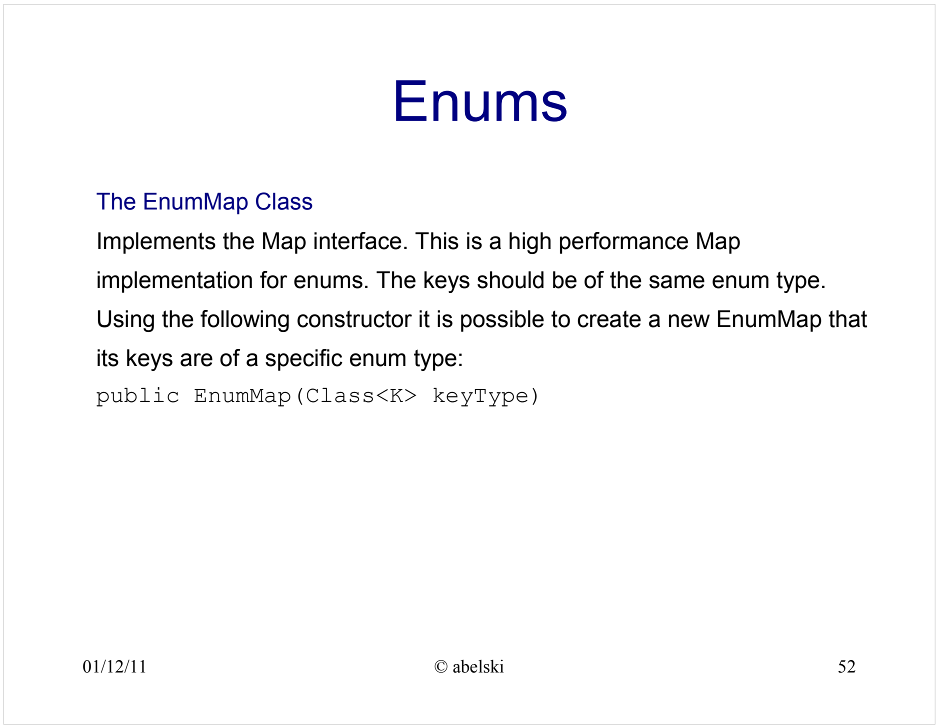
Enums

```
enum Day { SUN, MON, TUE, WED, THU, FRI, SAT };
EnumSet<Day> subSet = EnumSet.range(Day.SUN, Day.THU);
for(Day day : subSet)
{
     System.out.println(day);
}
```

01/12/11 © abelski 51

Internally, the EnumSet object uses a bit vector. Using this
simple bit vector enables its high performance.

The complete source code of the above sample can be found
within the samples folder of this topic.



Enums

The EnumMap Class

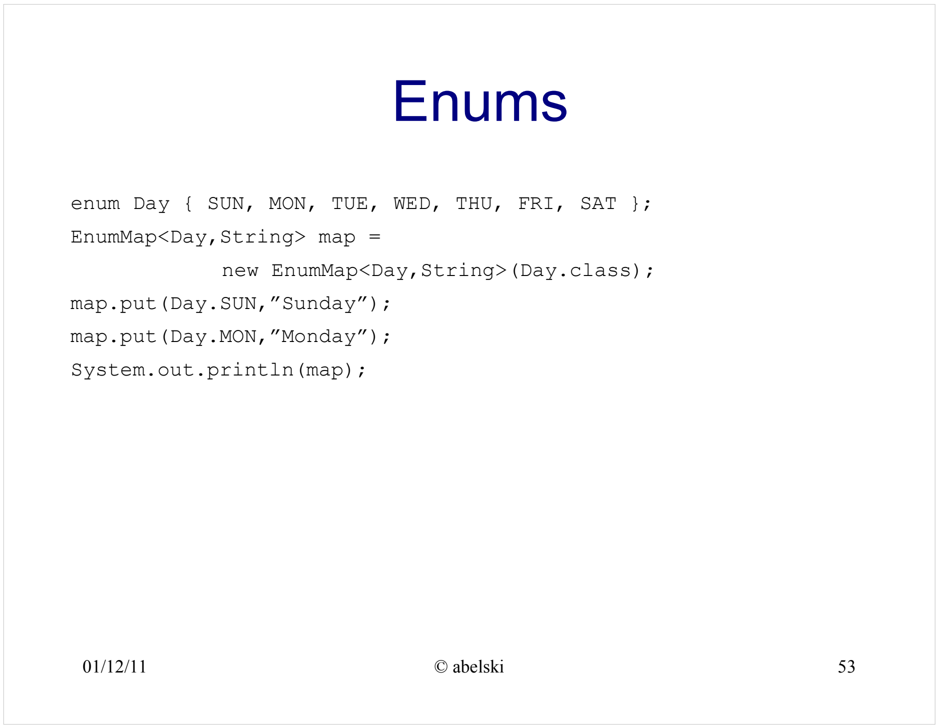
Implements the Map interface. This is a high performance Map implementation for enums. The keys should be of the same enum type. Using the following constructor it is possible to create a new EnumMap that its keys are of a specific enum type:

```
public EnumMap(Class<K> keyType)
```

01/12/11 © abelski 52

Internally, the EnumMap object uses a bit vector. Using this
simple bit vector enables its high performance.

The complete source code of the above sample can be found
within the samples folder of this topic.



Enums

```
enum Day { SUN, MON, TUE, WED, THU, FRI, SAT };
EnumMap<Day,String> map =
            new EnumMap<Day,String>(Day.class);
map.put(Day.SUN,"Sunday");
map.put(Day.MON,"Monday");
System.out.println(map);
```

01/12/11 © abelski 53

Internally, the EnumMap object uses a bit vector. Using this
simple bit vector enables its high performance.

The complete source code of the above sample can be found
within the samples folder of this topic.