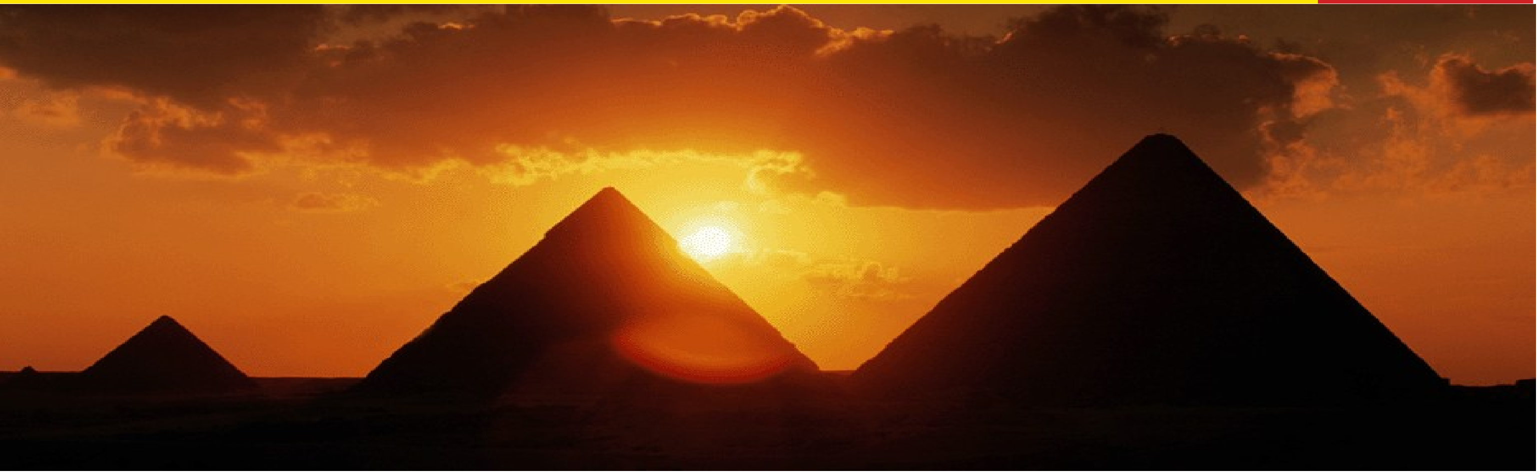
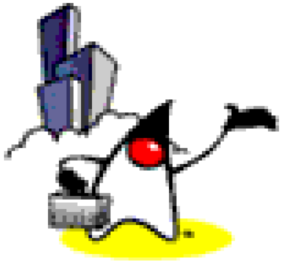


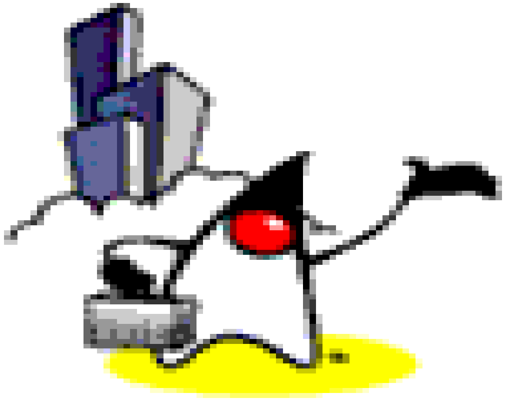
Creating Your Own Classes



Objectives

At the end of the lesson, the student should be able to:

- Create their own classes
- Declare properties (fields) and methods for their classes
- Use the this reference to access instance data
- Create and call overloaded methods
- Use access modifiers to control access to class members



Defining Your Own Class

Defining your own classes

- Things to take note of for the syntax defined in this section:

* means that there may be 0 or more occurrences of the line where it was applied to.

<description> indicates that you have to substitute an actual value for this part instead of typing it as it is.

[ ] indicates that this part is optional

Defining your own classes

- To define a class, we write:

```
<modifier> class <name> {
      <attributeDeclaration>*
      <constructorDeclaration>*
      <methodDeclaration>*
}
```

  - where
    - <modifier> is an access modifier, which may be combined with other types of modifier.

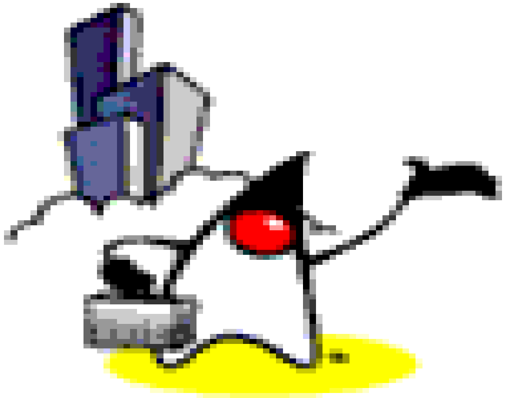
Example

```
public class StudentRecord {
    //we'll add more code here later

}
```

– where,
  - public - means that our class is accessible to other classes outside the package
  - class - this is the keyword used to create a class in Java
  - StudentRecord - a unique identifier that describes our class

Coding Guidelines

- Think of an appropriate name for your class. Don't just call your class XYZ or any random names you can think of.
- Class names starts with a CAPITAL letter - not a requirement, however.
- The filename of your class must have the SAME NAME as your class name.



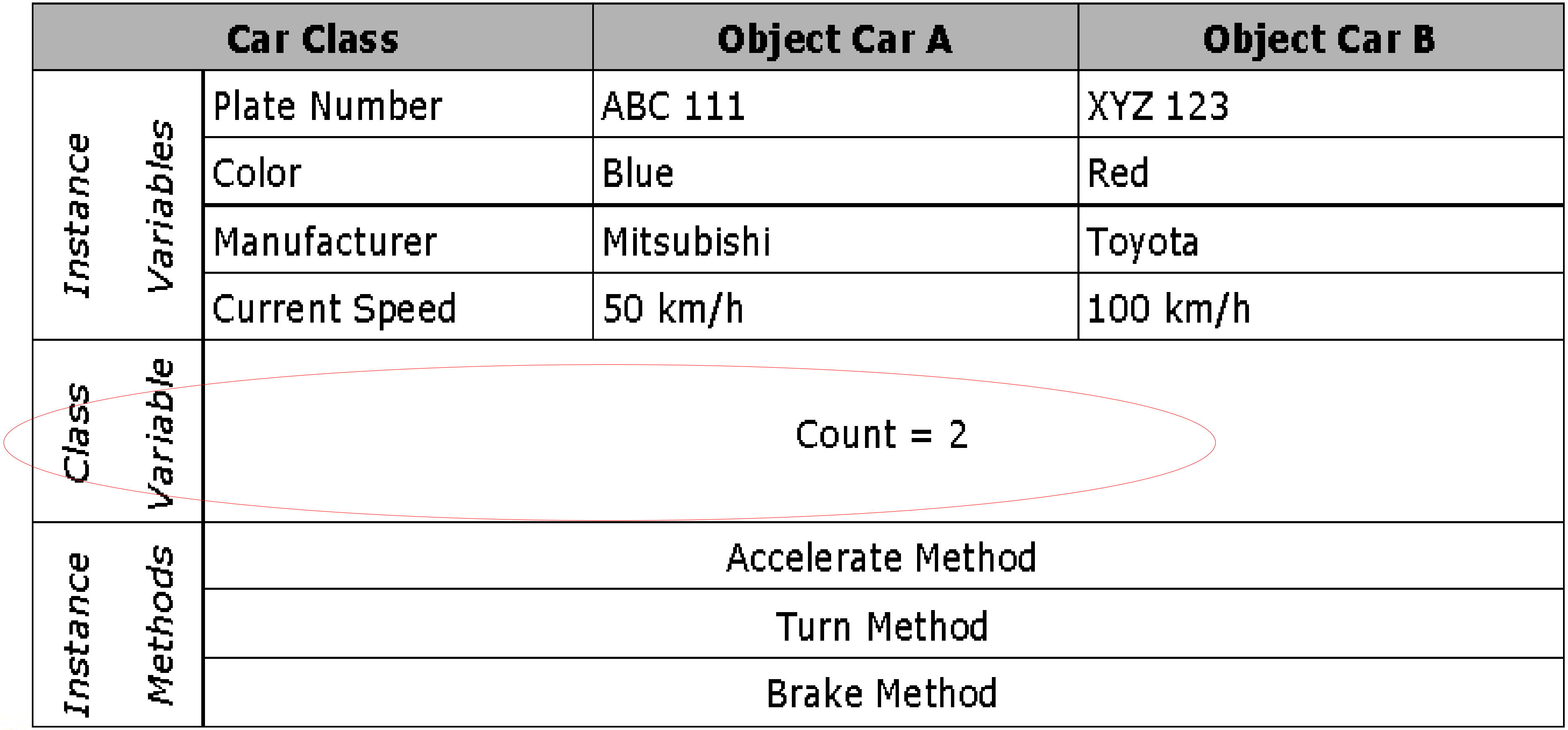
Instance Variables vs. Static Variables

Instance Variables (Properties) vs. Class (Static) Variables

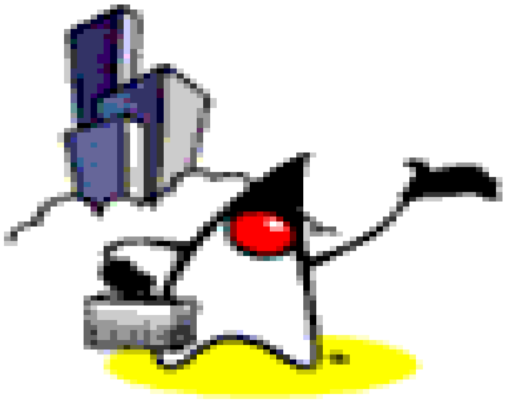
- Instance Variables
  - Belongs to an object instance
  - Value of variable of an object instance is different from the ones of other object object instances
- Class Variables (also called static member variables)
  - variables that belong to the whole class.
  - This means that they have the same value for all the object instances in the same class.

Class Variables

- For example,



Car Class		Object Car A	Object Car B
Instance Variables	Plate Number	ABC 111	XYZ 123
	Color	Blue	Red
	Manufacturer	Mitsubishi	Toyota
	Current Speed	50 km/h	100 km/h
Class Variable	Count = 2		
Instance Methods	Accelerate Method		
	Turn Method		
	Brake Method		



Instance Variables

Declaring Properties (Attributes)

- To declare a certain attribute for our class, we write,

```
<modifier> <type> <name> [=
  <default_value>];
```

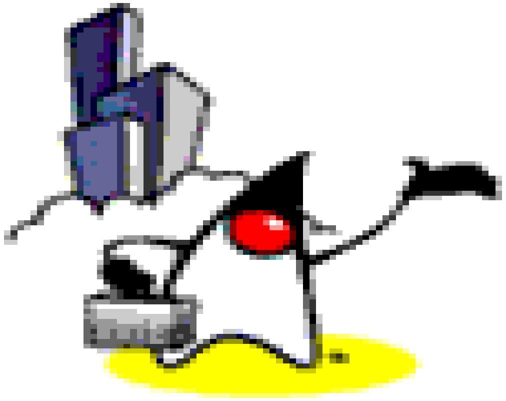
Instance Variables

```
public class StudentRecord {
    // Instance variables
    private String    name;
    private String    address;
    private int    age;
    private double mathGrade;
    private double englishGrade;
    private double scienceGrade;
    private double average;
    //we'll add more code here later
}
```

- where,
  - private here means that the variables are only accessible within the class. Other objects cannot access these variables directly. We will cover more about accessibility later.

Coding Guidelines

- Declare all your instance variables right after “public class Myclass {“
- Declare one variable for each line.
- Instance variables, like any other variables should start with a SMALL letter.
- Use an appropriate data type for each variable you declare.
- Declare instance variables as private so that only class methods can access them directly.
  - Encaptulation

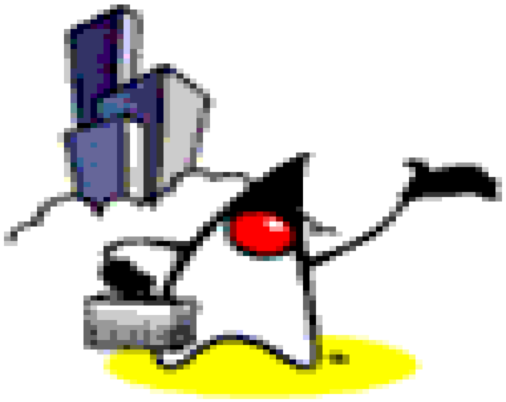


Static Variables

Class (static) variables

```
public class StudentRecord {
    //static variables we have declared
    private static int studentCount;
    //we'll add more code here later
}
```

– we use the keyword static to indicate that a variable is a static variable.



Methods

Declaring Methods

- To declare methods we write,

```
<modifier> <returnType>
  <name>(<parameter>*) {

        <statement>*

}
```

  - where,
    - <modifier> can carry a number of different modifiers
    - <returnType> can be any data type (including void)
    - <name> can be any valid identifier
    - <parameter> ::= <parameter_type> <parameter_name>[,]

Accessor (Getter) Methods

- Accessor methods
  - used to read values from our class variables (instance/static).
  - usually written as:

```
get<NameOfInstanceVariable>
```

  - It also returns a value.

Example 1: Accessor (Getter) Method

```
public class StudentRecord {
    private String  name;
    :
    public String getName(){
       return name;
    }
}
```

- where,
  - public - means that the method can be called from objects outside the class
  - String - is the return type of the method. This means that the method should return a value of type String
  - getName - the name of the method
  - () - this means that our method does not have any parameters

Example 2: Accessor (Getter) Method

```
public class StudentRecord {
    private String  name;
    // some code

    // An example in which the business logic is
    // used to return a value on an accessor method
    public double getAverage(){
       double result = 0;
       result=(mathGrade+englishGrade+scienceGrade)/3;
       return result;
    }
}
```

Mutator (Setter) Methods

- Mutator Methods
  - used to write or change values of our class variables (instance/static).
  - Usually written as:

```
set<NameOfInstanceVariable>
```

Example: Mutator (Setter) Method

```
public class StudentRecord {
    private String  name;
    :
    public void setName( String temp ){
        name = temp;
    }
}
```

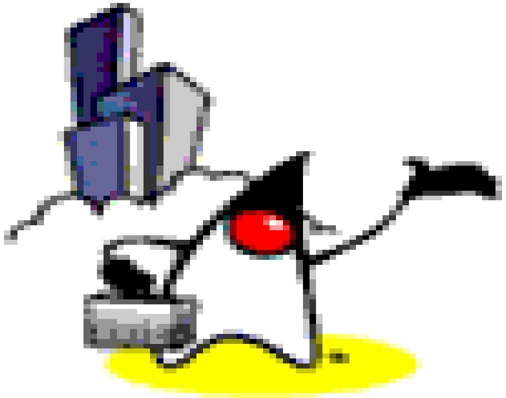
- where,
  - public - means that the method can be called from objects outside the class
  - void - means that the method does not return any value
  - setName - the name of the method
  - (String temp) - parameter that will be used inside our method

Multiple return statements

- You can have multiple return statements for a method as long as they are not on the same block.
- You can also use constants to return values instead of variables.

Example: Multiple return statements

```
public String getNumberInWords( int num ){
    String defaultNum = "zero";
    if( num == 1 ){
        return "one"; //return a constant
    }
    else if( num == 2){
        return "two"; //return a constant
    }
    //return a variable
    return defaultNum;
}
```



Static Methods

Static methods

```
public class StudentRecord {
    private static int studentCount;
    public static int getStudentCount(){
        return studentCount;
    }
}
```

- where,
  - public- means that the method can be called from objects outside the class
  - static-means that the method is static and should be called by typing,[ClassName].[methodName]. For example, in this case, we call the method `StudentRecord.getStudentCount()`
  - int- is the return type of the method. This means that the method should return a value of type int
  - getStudentCount- the name of the method
  - ()- this means that our method does not have any parameters

Coding Guidelines

- Method names should start with a SMALL letter.
- Method names should be verbs
- Always provide documentation before the declaration of the method. You can use Javadocs style for this. Please see example.

When to Define Static Method?

- When the logic and state does not involve specific object instance
  - Computation method
  - add(int x, int y) method
- When the logic is a convenience without creating an object instance
  - Integer.parseInt();

Source Code for StudentRecord class

```
public class StudentRecord {

   // Instance variables
    private String    name;
    private String    address;
    private int    age;
    private double mathGrade;
    private double englishGrade;
    private double scienceGrade;
    private double average;
    private static int studentCount;
```

Source Code for StudentRecord Class

```
/**
 * Returns the name of the student (Accessor method)
 */
public String getName(){
    return name;
}


/**
 * Changes the name of the student (Mutator method)
 */
public void setName( String temp ){
    name = temp;
}
```

Source Code for StudentRecord Class

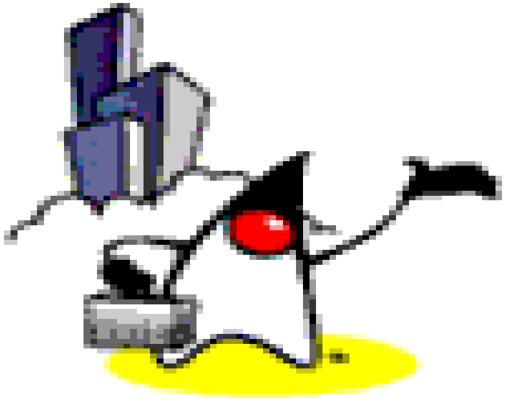
```
/**
 * Computes the average of the english, math and science
 * grades (Accessor method)
 */
public double getAverage(){
    double result = 0;
    result = ( mathGrade+englishGrade+scienceGrade )/3;
    return result;
}
/**
 * returns the number of instances of StudentRecords
 * (Accessor method)
 */
public static int getStudentCount(){
    return studentCount;
}
```

Sample Source Code that uses StudentRecord Class

```
public class StudentRecordExample
{
   public static void main( String[] args ){

    //create three objects for Student record
    StudentRecord   annaRecord = new StudentRecord();
    StudentRecord   beahRecord = new StudentRecord();
    StudentRecord   crisRecord = new StudentRecord();

    //set the name of the students
    annaRecord.setName("Anna");
    beahRecord.setName("Beah");
    crisRecord.setName("Cris");

    //print anna's name
    System.out.println( annaRecord.getName() );

    //print number of students
    System.out.println("Count="+StudentRecord.getStudentCount());
   }
}
```

Program Output

```
Anna
Student Count = 0
```



Overloading Methods

Overloading Methods

- Method overloading
  - allows a method with the same name but different parameters, to have different implementations and return values of different types
  - can be used when the same operation has different implementations.

- Always remember that overloaded methods have the following properties:
  - the same method name
  - different parameters or different number of parameters
  - return types can be different or the same

Example

```
public void print( String temp ){
   System.out.println("Name:" + name);
   System.out.println("Address:" + address);
   System.out.println("Age:" + age);
}

public void print(double eGrade, double mGrade,
                              double sGrade)
   System.out.println("Name:" + name);
   System.out.println("Math Grade:" + mGrade);
   System.out.println("English Grade:" + eGrade);
   System.out.println("Science Grade:" + sGrade);
}
```

Example

```
public static void main( String[] args )
{
    StudentRecord  annaRecord = new StudentRecord();

    annaRecord.setName("Anna");
    annaRecord.setAddress("Philippines");
    annaRecord.setAge(15);
    annaRecord.setMathGrade(80);
    annaRecord.setEnglishGrade(95.5);
    annaRecord.setScienceGrade(100);

    //overloaded methods
    annaRecord.print( annaRecord.getName() );
    annaRecord.print( annaRecord.getEnglishGrade(),
            annaRecord.getMathGrade(),
                annaRecord.getScienceGrade());
}
```

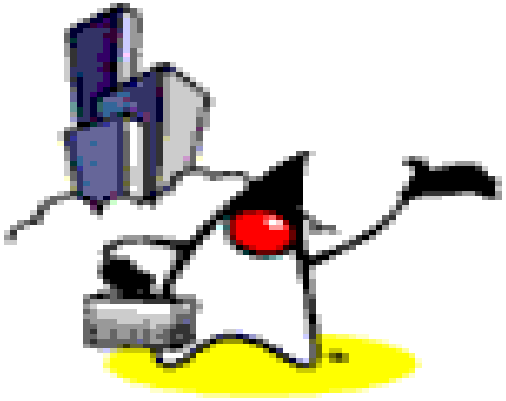
Output

- we will have the output for the first call to print,

```
Name:Anna
Address:Philippines
Age:15
```

- we will have the output for the second call to print,

```
Name:Anna
Math Grade:80.0
English Grade:95.5
Science Grade:100.0
```



Constructors (Constructor Methods)

Constructors

- Constructors are important in instantiating an object. It is a method where all the initializations are placed.

- The following are the properties of a constructor:
  - Constructors have the same name as the class
  - A constructor is just like an ordinary method, however only the following information can be placed in the header of the constructor,
  - scope or accessibility identifier (like public...), constructor's name and parameters if it has any.
  - Constructors does not have any return value
  - You cannot call a constructor directly, it can only be called by using the new operator during class instantiation.

Constructors

- To declare a constructor, we write,

```
<modifier> <className> (<parameter>*) {
     <statement>*
}
```

Default Constructor (Method)

- The **default constructor (no-arg constructor)**
  - is the constructor without any parameters.
  - If the class does not specify any constructors, then an implicit default constructor is created.

Example: Default Constructor Method of StudentRecord Class

```
public StudentRecord()
{
    //some code here
}
```

Overloading Constructor Methods

```
public StudentRecord(){
    //some initialization code here
}
public StudentRecord(String temp){
    this.name = temp;
}
public StudentRecord(String name, String address){
    this.name = name;
    this.address = address;
}
public StudentRecord(double mGrade, double eGrade,
                     double sGrade){
    mathGrade = mGrade;
    englishGrade = eGrade;
    scienceGrade = sGrade;
}
```

Using Constructors

- To use these constructors, we have the following code,

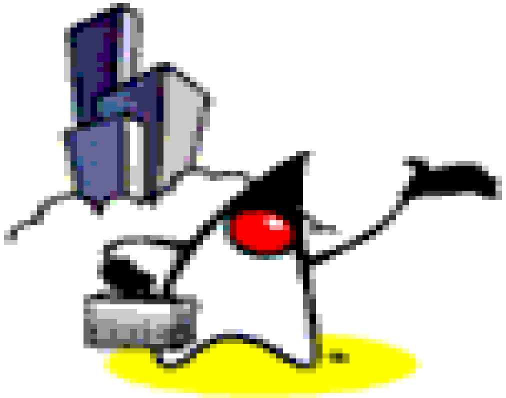
```
public static void main( String[] args ){
    //create three objects for Student record
    StudentRecord  annaRecord=new StudentRecord("Anna");
    StudentRecord  beahRecord=new StudentRecord("Beah",
                                    "Philippines");
    StudentRecord  crisRecord=new
      StudentRecord(80,90,100);
    //some code here
}
```

“this()” constructor call

- Constructor calls can be chained, meaning, you can call another constructor from inside another constructor.
- We use the this() call for this
- There are a few things to remember when using the this() constructor call:
  - When using the this constructor call, IT MUST OCCUR AS THE FIRST STATEMENT in a constructor
  - It can ONLY BE USED IN A CONSTRUCTOR DEFINITION. The this call can then be followed by any other relevant statements.

Example

```
public StudentRecord(){
    this("some string");

}

public StudentRecord(String temp){
    this.name = temp;
}

public static void main( String[] args )
{

    StudentRecord   annaRecord = new StudentRecord();
}
```



“this” Reference

“this” reference

- The this reference
  - refers to current object instance itself
  - used to access the instance variables shadowed by the parameters.

- To use the this reference, we type,

```
this.<nameOfTheInstanceVariable>
```

- You can only use the this reference for instance variables and NOT static or class variables.

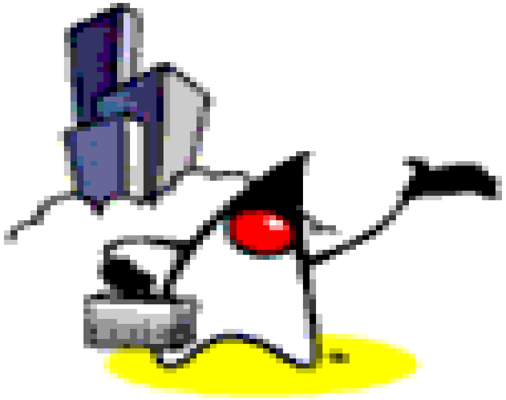
“this” reference

- The this reference is assumed when you call a method from the same object

```
public class MyClass {
        void aMethod() {
                // same thing as this.anotherMethod()
                anotherMethod();
        }
        void anotherMethod() {
            // method definition here...
        }
}
```

Example

```
public void setAge( int age ){
    this.age = age;
}
```



Access Modifiers

Access Modifiers

- There are four different types of member access modifiers in Java:
  - public (Least restrictive)
  - protected
  - default
  - private (Most restrictive)
- The first three access modifiers are explicitly written in the code to indicate the access type, for the fourth one which is default, no keyword is used.

public accessibility

- public access
  - specifies that class members (variables or methods) are accessible to anyone, both inside and outside the class and outside of the package.
  - Any object that interacts with the class can have access to the public members of the class.
  - Keyword: public

Example: “public” Access Modifer

```
public class StudentRecord {
    //default access to instance variable
    public int name;

    //default access to method
    public String getName(){
        return name;
    }
}
```

protected accessibility

- protected access
  - Specifies that the class members are accessible only to methods in that class and the subclasses of the class.
  - The subclass can be in different packages
  - Keyword: protected

Example: “protected” Access Modifier

```
public class StudentRecord {
    //default access to instance variable
    protected int name;

    //default access to method
    protected String getName(){
        return name;
    }
}
```

default accessibility

- Default access
  - specifies that only classes in the same package can have access to the class' variables and methods
  - no actual keyword for the default modifier; it is applied in the absence of an access modifier.

Example

```
public class StudentRecord {
    //default access to instance variable
    int  name;

    //default access to method
    String getName(){
       return name;
    }
}
```

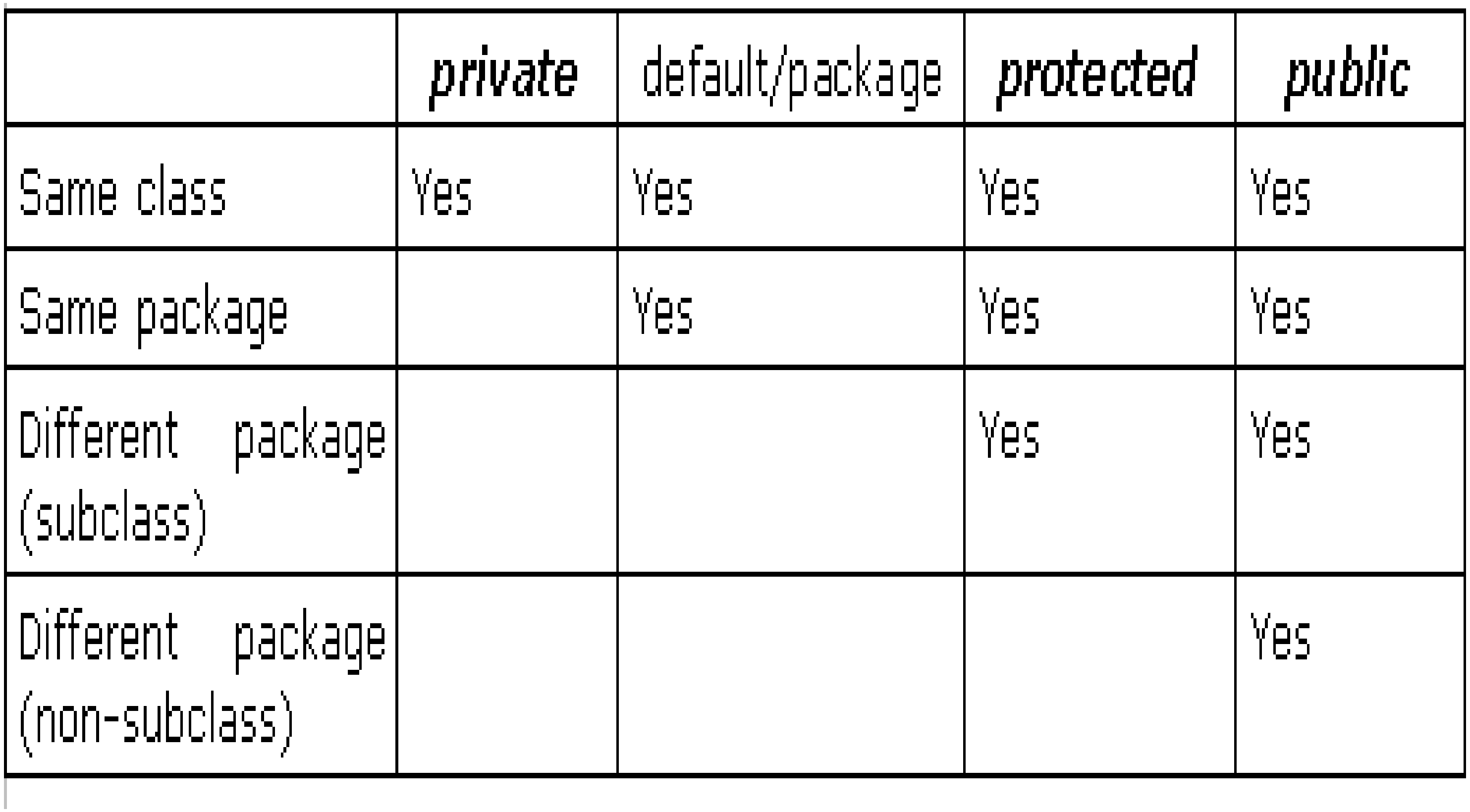
private accessibility

- private accessibility
  - specifies that the class members are only accessible by the class they are defined in.
  - Keyword: private

Example: “private” Access Modifier

```
public class StudentRecord {
    //default access to instance variable
    private int   name;

    //default access to method
    private String getName(){
       return name;
    }
}
```

Java Program Structure: The Access Modifiers



	private	default/package	*protected*	*public*
Same class	Yes	Yes	Yes	Yes
Same package		Yes	Yes	Yes
Different package (subclass)			Yes	Yes
Different package (non-subclass)				Yes

Coding Guidelines

- The instance variables of a class should normally be declared private, and the class will just provide accessor and mutator methods to these variables.

Summary

- Defining your own classes
- Declaring Fields (instance, static/class)
- Declaring Methods (accessor, mutator, static)
- Returning values and Multiple return statements
- The this reference
- Overloading Methods
- Constructors (default, overloading, this() call)
- Packages
- Access Modifiers (default, public, private, protected)