Java Object Oriented Programming

What is it?

Sat Mar 22
03:57:53
2025

What is the Java API

API stands for Application Programming Interface

The API is a large collection of ready-made software components that provide many useful capabilities.

It has an index or collection of all Java packages, classes and interfaces, with all of their methods, fields and constructors, and how to use them.

It is grouped into libraries of related **classes** and interfaces; these libraries are known as ***packages.***

Where's the API?

The latest version of java is 7. You can find the current version here .

http://docs.oracle.com/javase/7/docs/api/

docs.oracle.com/javase/7/docs/api/

Most Visited | Getting Started | Sign In | Sign In | Teach ICT - Greenfoot ... | Adobe Connect Centr... | UAlbany ICSI310 Sprin... | YouTube to mp3 Conv... | Please Login | Google: Exploring Co... | MOOC Local

Java™ Platform Standard Ed. 7

All Classes

Packages

java.applet
java.awt
java.awt.color
java.awt.datatransfer
java.awt.dnd
java.awt.event

All Classes

AbstractAction
AbstractAnnotationValueVisitor6
AbstractAnnotationValueVisitor7
AbstractBorder
AbstractButton
AbstractCellEditor
AbstractCollection
AbstractColorChooserPanel
AbstractDocument
AbstractDocument.AttributeContext
AbstractDocument.Content
AbstractDocument.ElementEdit
AbstractElementVisitor6
AbstractElementVisitor7
AbstractExecutorService
AbstractInterruptibleChannel
AbstractLayoutCache
AbstractLayoutCache.NodeDimensions
AbstractList
AbstractListModel
AbstractMap
AbstractMap.SimpleEntry
AbstractMap.SimpleImmutableEntry

Overview | Package | Class | Use | Tree | Deprecated | Index | Help

Prev | Next | Frames | No Frames

Java™ Platform, Standard Edition 7
API Specification

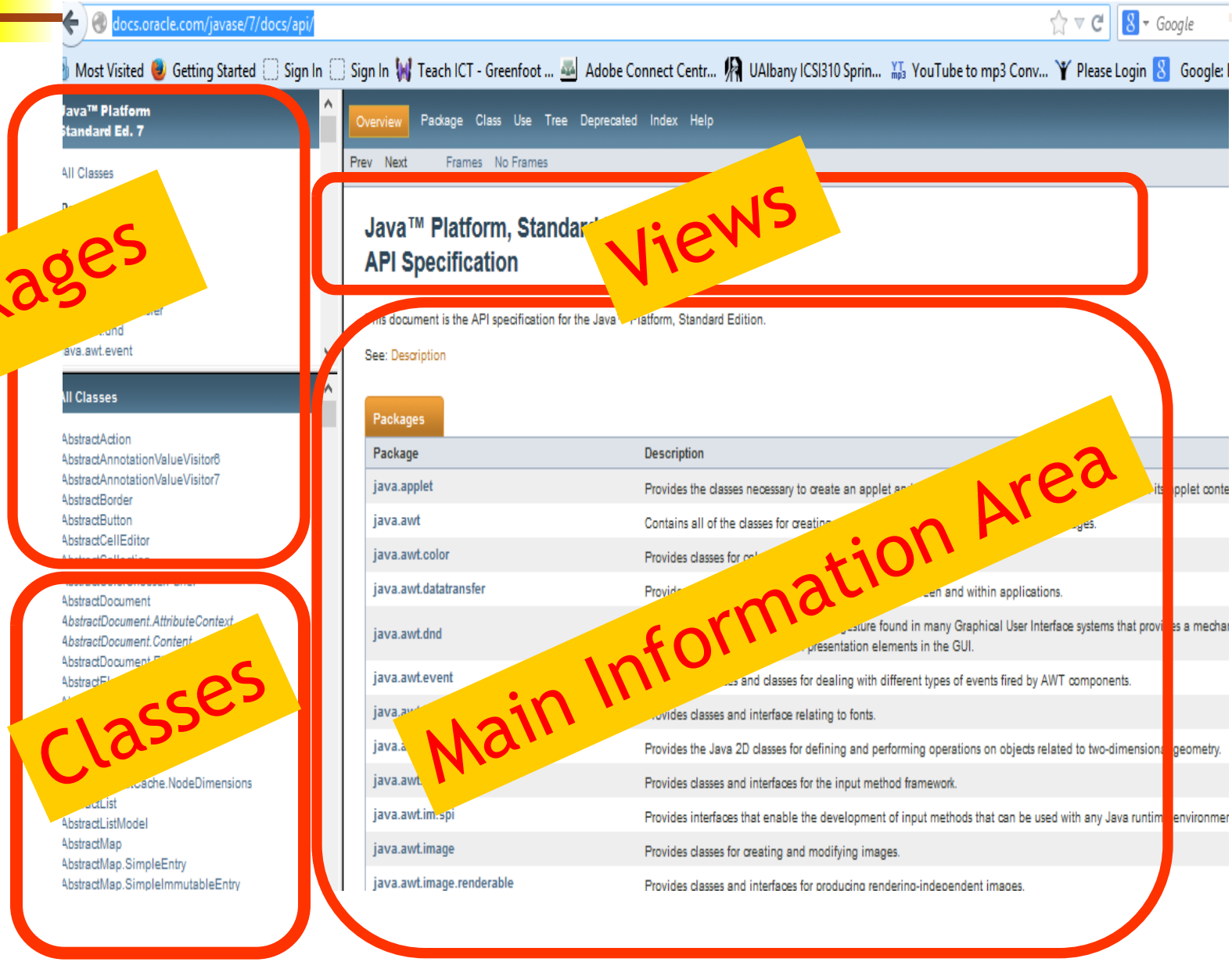
This document is the API specification for the Java™ Platform, Standard Edition.

See: Description

Packages

Package	Description
java.applet	Provides the classes necessary to create an applet and the classes an applet uses to communicate with its applet context.
java.awt	Contains all of the classes for creating user interfaces and for painting graphics and images.
java.awt.color	Provides classes for color spaces.
java.awt.datatransfer	Provides interfaces and classes for transferring data between and within applications.
java.awt.dnd	Drag and Drop is a direct manipulation gesture found in many Graphical User Interface systems that provides a mechanism to transfer information between two entities logically associated with presentation elements in the GUI.
java.awt.event	Provides interfaces and classes for dealing with different types of events fired by AWT components.
java.awt.font	Provides classes and interface relating to fonts.
java.awt.geom	Provides the Java 2D classes for defining and performing operations on objects related to two-dimensional geometry.
java.awt.im	Provides classes and interfaces for the input method framework.
java.awt.im.spi	Provides interfaces that enable the development of input methods that can be used with any Java runtime environment.
java.awt.image	Provides classes for creating and modifying images.
java.awt.image.renderable	Provides classes and interfaces for producing rendering-independent images.

Overview


docs.oracle.com/javase/7/docs/api/
Most Visited
Getting Started
Sign In
Sign In
Teach ICT - Greenfoot ...
Adobe Connect Centr...
UAlbany ICSI310 Sprin...
YouTube to mp3 Conv...
Please Login
Google
Overview
Package
Class
Use
Tree
Deprecated
Index
Help
Prev
Next
Frames
No Frames
Java™ Platform, Standard Ed. 7
All Classes
Java™ Platform, Standard
API Specification
See: Description
Packages
Package
Description
java.applet
java.awt
java.awt.color
java.awt.datatransfer
java.awt.dnd
java.awt.event
java.awt.image
java.awt.image.renderable
AbstractAction
AbstractAnnotationValueVisitor6
AbstractAnnotationValueVisitor7
AbstractBorder
AbstractButton
AbstractCellEditor
AbstractDocument
AbstractList
AbstractListModel
AbstractMap
AbstractMap.SimpleEntry
AbstractMap.SimpleImmutableEntry
Packages
Views
Classes
Main Information Area

General approach

- If you know the name of the package, click it in the upper left panel; or click All Classes
- Click on the class in the lower left panel
- Scroll in the right pane to find the summary of the field, method, or constructor you want
  - Or just read the general description
- For more information, click the link in the summary to go to the detailed information

The Packages panel

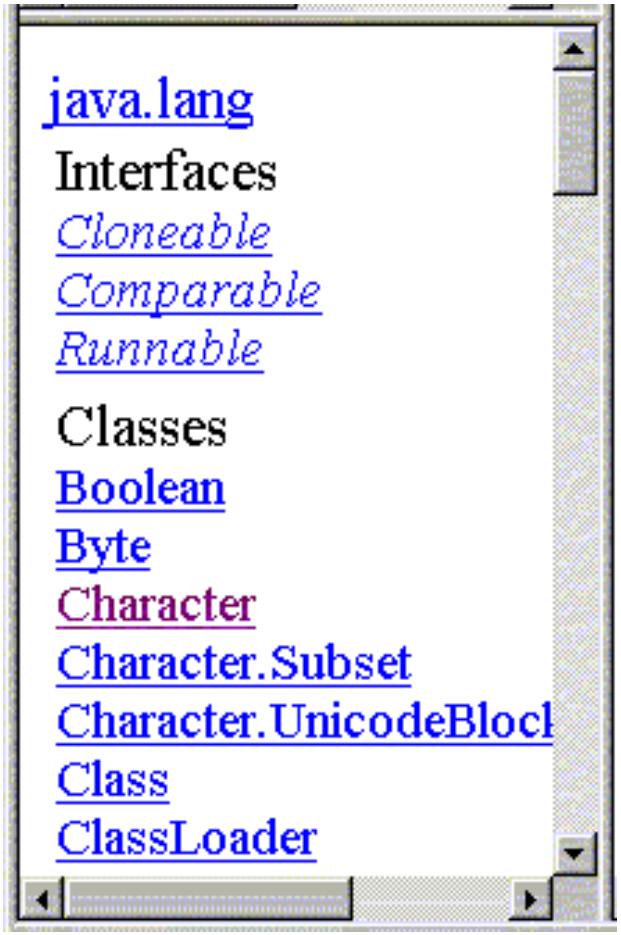
- Choose the package you are interested in
- Or, choose All Classes
- Classes in java.lang are automatically imported into every program--you don't have to do it yourself

What Is a Package?

A package is a namespace that organizes a set of related classes and interfaces. You must import the package into your program for the class that you want to use.

import java.util.*;	This will import the entire java.util package.
import java.util.ArrayList;	This will import the ArrayList class so you can use it.

The Classes panel



- This panel shows both classes and interfaces
- We haven't yet talked about interfaces
- Note that some classes have names *similar to* primitive types (Boolean, Byte, Character). These are the wrapper classes used with ArrayList.

Random Class

If you scroll down the classes you will find Random. Click and it will bring up the API information for the Random Class.

Java™ Platform Standard Ed. 7

All Classes

Packages

java.applet
java.awt

QueuedJobCount
RadialGradientPaint
Random
RandomAccess
RandomAccessFile
Raster
RasterFormatException
RasterOp
RC2ParameterSpec
RC5ParameterSpec
Rdn
Readable
ReadableByteChannel
Reader
ReadOnlyBufferException
ReadOnlyFileSystemException
ReadPendingException

Overview Package Class Use Tree Deprecated Index Help

Prev Class Next Class Frames No Frames

Summary: Nested | Field | Constr | Method Detail: Field | Constr | Method

java.util

Class Random

java.lang.Object
    java.util.Random

All Implemented Interfaces:

Serializable

Direct Known Subclasses:

SecureRandom, ThreadLocalRandom

```
public class Random
extends Object
implements Serializable
```

The random class is in the java.util package.

You would import that package to use the class.

Boolean Class

Scroll and find the Boolean class. Click on it to bring up the API information.

Most Visited | Getting Started | Sign In | Sign In | Teach ICT - Greenfoot ... | Adobe Connect Centr... | UAlbany ICSI310 Sprin... | YouTube to mp3 Conv... | Please Login | Google: Exploring Co...

java.awt.print
java.beans
java.beans.beancontext
java.io
java.lang
java.lang.annotation
java.lang.instrument
java.lang.invoke
java.lang.management
java.lang.ref
java.lang.reflect
java.math
java.net
java.nio
java.nio.channels

Packages

Interfaces

Appendable
AutoCloseable
CharSequence
Cloneable
Comparable
Iterable
Readable
Runnable
Thread.UncaughtExceptionHandler

Classes

Boolean
Byte
Character
Character.Subset
Character.UnicodeBlock
Class
ClassLoader
ClassValue
Compiler
Double
Enum
Float
InheritableThreadLocal
Integer
Long
Math
Number
Object
Package
Process

Overview | Package | Class | Use | Tree | Deprecated | Index | Help

Prev Class | Next Class | Frames | No Frames

Summary: Nested | Field | Constr | Method Detail: Field | Constr | Method

java.lang

Class Boolean

java.lang.Object
  java.lang.Boolean

All Implemented Interfaces:

Serializable, Comparable<Boolean>

```
public final class Boolean
extends Object
implements Serializable, Comparable<Boolean>
```

The Boolean class wraps a value of the primitive type `boolean` in an object. An object of type `Boolean` contains a single field whose type is boo

In addition, this class provides many methods for converting a `boolean` to a `String` and a `String` to a `boolean`, as well as other constants

Since:

JDK1.0

See Also:

Serialized Form

Field Summary

Fields

Modifier and Type	Field and Description
`static Boolean`	`FALSE` The `Boolean` object corresponding to the primitive value `false`.
`static Boolean`	`TRUE` The `Boolean` object corresponding to the primitive value `true`.

The Boolean class is in the lang. package.

You do not have to use an import statement for any of the classes in the lang. package.

The main information area for classes

- General description of the class
- Field summary
- Constructor summary
- Method summary
- Field detail
- Constructor detail
- Method detail

- In each case, the "summary" is the *first sentence* of the "detail"

General Summary of class

The summary section gives information about the class

Class Boolean

java.lang.Object
    java.lang.Boolean

All Implemented Interfaces:

Serializable, Comparable<Boolean>

```
public final class Boolean
extends Object
implements Serializable, Comparable<Boolean>
```

The Boolean class wraps a value of the primitive type `boolean` in an object. An object of type `Boolean` contains a single field whose type is `boolean`.

In addition, this class provides many methods for converting a `boolean` to a `String` and a `String` to a `boolean`, as well as other constants and methods useful when dealing with a `boolean`.

Since:

JDK1.0

See Also:

Serialized Form

Field Summary Information

Serialized Form

Field Summary

Fields

Modifier and Type	Field and Description
`static Boolean`	`FALSE` The `Boolean` object corresponding to the primitive value `false`.
`static Boolean`	`TRUE` The `Boolean` object corresponding to the primitive value `true`.
`static Class<Boolean>`	`TYPE` The Class object representing the primitive type boolean.

A field is an attribute. A field may be a class's variable, an object's variable. They are available to the class if it is a class variable or any object created if it is an instance variable.

These would be global variables. They are seen throughout the class. Variables created inside a method can only be seen inside a method. They are local variables.

Constructors

A constructor has two purposes:

- A constructor is special method that creates an object. When an object is created it calls the constructor for the class.
- Why have you been able to create objects before? Java has a default constructor the class that is called if you do not create one. The default constructor requires on parameter information when created. Once you create a constructor; however, you lose the default constructor.

Constructor

Constructor Summary

Constructors

Constructor and Description
`Boolean(boolean value)` Allocates a `Boolean` object representing the `value` argument.
`Boolean(String s)` Allocates a `Boolean` object representing the value `true` if the string argument is not `null` and is equal, ignoring case, to the string `"true"`.

Let's look at the first constructor for the class.

public Boolean(boolean value)

You would create an object like this:

Boolean b1 = new Boolean(true); or

Boolean b2 = new Boolean(false);

Using the API

If I wanted to know how to create a object from the Random class I would go to the Random class in the Java API.
http://docs.oracle.com/javase/7/docs/api/java/util/Random.html

I would look at the information for the constructor to see how to create the object.
Random **generator** = new Random();

Constructor Summary

Constructors	
Constructor and Description	
`Random()`	Creates a new random number generator.

Now I would look at the methods that are Available to use and find the one I need. For example to create a random integer between 0 and a specified number I would use this method.

`nextInt``(int n)`

Returns a pseudorandom, uniformly distributed `int` value between 0 (inclusive) and the specified value (exclusive), drawn from this random number generator's sequence.

The object you created calls the methods from its class.

int randomNumber = generator.nextInt(10);

This code produces a random number between 0 and n-1. (0 to 9) and the result will be stored in the variable randomNumber.

Object Oriented Programming

- Java is an object oriented program.

Class Structure

Import Statement

Class name line 7

Field Data
Line 12, 13, 14, 15

Constructor
There are 2 constructors
One is overloaded constructor.

```
1   //import statements go first
2   import java.util.*;
3   import java.util.ArrayList;
4
5   //Class name goes next
6
7   public class Student
8   {
9
10   //Field Data.  Every object created gets a copy.  They are created private to protect the data
11
12   private String name;
13   private int age;
14   private int grade;
15   private final static String school = "My High School";
16
17  /*Constructor goes next.  Constructor specifies how to create the object of the class.
18   * It is a method that doesn't return anything. Neither void nor a return method. It creates the object and will
19   * initialize the variables in the field data section.
20   * */
21
22   public Student()
23   {
24    name = "Student";
25    age = 0;
26    grade = 0;
27   }
28
29   public Student(String n, int a, int g)
30   {
31    name = n;
32    age = a;
33    grade = g;
34   }
```

Any classes you need to use must be imported except the Java.lang package.

Instance variables, final variables, class variables go in this area.

You would create a Student object like this:
Student s1= new Student(); Each student created this way would have the name "Student" age 0 and grade 0

You would create a Student object like this:
Student s2 = new Student("Sally Smith", 17, 12);
You must supply the name of the student, the age and the grade in that order.

Field Data

TYPES OF VARIABLES

<u>Global</u> – Can be seen throughout the program

<u>Local</u> – used in a method and can only be seen in that method.

- **Instance Variable** – private and every
- object created receives its own copy.

```
12  private String name;
13  private int age;
14  private int grade;
```

- **Final variable** – Cannot be changed usually created static as well. Usually static as well.
- **Static variable**: one copy for the entire class. Every object gets a copy but the same copy.

```
15  private final static String school = "My High School";
```

Creating Variables

Instance variables: Declared in the Field area

- Instance variables are declared in a class, but outside a method, constructor or any block.

- Declared private

- Encapsulation / information hiding (you cannot directly access the variable from another class. You create methods that will change and return the information from the variables.

Final Variables

- Variable declared as final cannot be changed. It is a constant.
- The word final is used and the variable is in all caps.

```
private final int PI = 3.14;
private final int PCENT = .07;
```

Final variables are usually declared static as well.

```
private static final int PI = 3.14;
```

Static Variables

- There would only be one copy of each class variable per class, regardless of how many objects are created from it.

- Static variables are rarely used other than being declared as constants. Constants are variables that are private, final and static. Constant variables never change from their initial value.

private static int number = 0; // this variable will can change but every object uses the same copy of the variable. All objects have the same number

Constructor

Purpose and function:

•Constructors have one purpose in life: to create an instance of a class.

•A constructor consist of the following format:

visibility type	**Class Name**	**(parameter info)**
public	Student	();

The above is an example of a default constructor.

•Constructors are created with the **visibility public**. Otherwise you could not create an object outside of the class.

•If you do not create a constructor for the class, Java supplies a default constructor for you.

CONSTRUCTOR

Constructors also initialize the instance variables for each object when the object is created. Each object gets a copy of the field data when the object is created.

Instance Variables
created in the field area:

```
12    private String name;
13    private int age;
14    private int grade;
```

```
public Student()
{
  name = "Student";
  age = 0;
  grade = 0;
}
```

Student s1 = new Student();
// using the above constructor

Every student created would have a name “Student” age 0 and grade 0
Every student would receive a copy of the final variable school “My High School”

OVERLOADED CONSTRUCTORS WITH PARAMETER INFORMATION

Constructors can also pass information to the object through the parameter that will initialize the field data. When the object is created the information must be passed in the same order as it is listed in the parameter of the constructor

```
public Student(String n, int a, int g)
{
name = n;
age = a;
grade = g;
}
```

This allows each student to be given unique information when it is created.

You would create an object like this:

Student s2= new Student("Jim", 17, 12);

(String n , int a, String g)

Student s2 name is Jim, age is 17 and grade is 12

Overloaded constructors

Constructors with parameter information

```
public Student(String name, int age, int grade)
{
name = name;
age = age;
grade = grade;
}
```

Can't do this!

```
public Student(String name, int age, int grade)
{
this.name = name;
this.age = age;
this.grade = grade;
}
```

Can do this

this reference

If you have the same variable name in the parameter as the instance variable name **you must distinguish between the two.**

The keyword **this** refers to the object.

- This objects color is color.
- This objects legs are legs
- This objects name is name

Accessing information from the field data

- Instance variables are created private. Therefore you cannot access them directly from outside the class. If you need to get the information or change the information you must create accessor and mutator method.

- You create methods called accessors and **mutators** to get the information and set the information for the instance variables.

Information Hiding

Information about an object is hidden to protect the data. If others could change your objects data directly the data would not be protected. This is why they are created private.

Methods are created to change (set) and get (return) the information from the object. These are public.

Accessor Method

Accessor Methods: Returns information from the instance variables. The information returned can be a primitive data type or an object. Accessor methods are always a return type method. (not void) They are called getter methods

<u>FORMAT:</u>

They are called getter methods

visibility **returnType get**Variable() {

```
public  String getName()
{
return name;
}
```

This method returns the information from the instance variable name

Mutator Methods

Methods created to change the data in the instance variables. The information you want to change the variable to is passed through the parameter. **They are void. Must use the variable passed through the parameter to set the information.**

```
public void setName(String n)
{
name = n;
}

public int setAge(int a)
{
age = a;
}

public int setGrade(int g)
{
grade = g;
}
```

Called setter methods.

Set data in the instance variables.

```
//Add getter (accessor) and setter (mutator) for each
 //instance variable.

 public void setName(String n)
{
  name = n;
}
public String getName()
{
  return name;
}
public void setAge(int a)
 {
   age = a;
 }
 public int getAge()
 {
   return age;
 }
 public void setGrade(int g)
{
  grade = g;
}
public int getGrade()
{
  return grade;
}
```

Accessor Methods to return the data stored in the instance variables. It is the only way to get information from the variables. You must create an accessor method.

They are usually named get and then the variable they are returning information from. They are always the return type of the variable.

Mutator Methods:
Called setter methods. They allow you to change the information in the instance variables.

Set data in the instance variables.

Acessor Methods: red

Mutator methods: black

toString() method

String	toString() Returns a string representation of the object.

The toString method comes from the Object class. The Object class in the highest hierarchy in Java. All classes are children of the Object class.

To print the information from an object you need to over ride the toString method (write your own toString method for the class).

If you do not override the toString method it will print out a hashtag representation of the object. Student@e6dd170

toString()

- It is a return method that returns whatever information you want about the object.

```
public String toString()
{
 return "Student Name:  " + getName() + "  age: " + getAge() + "  grade: " + getGrade() +"   " +Student.school;


}
```

- This would return the information from the methods getname(), getAge(), getGrade() and the static class variable school.

- Remember the school variable is static and must called by the class name.

```
Student Name:  Student  age: 0  grade: 0   My High School
Student Name:  Joe Johnson  age: 17  grade: 11   My High School
>
```

Calling the methods and creating objects in the main

Usually the class doesn't contain a main method. It isn't very functional if it does.

You create a driver class that contains the main. You create an object of the class and it calls its methods.

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

  Student s1 = new Student();

   System.out.println(s1.toString());

  s1.setName("Joe Johnson");
  s1.setAge(17);
  s1.setGrade(11);

  System.out.println(s1.toString());

  Student s2 = new Student("Lucy Lee", 16, 10);
  System.out.println(s2.toString());

 }
}
```

OUTPUT

```
Welcome to DrJava. Working directory is F:\School year 2013 2014\AP Computer Science
> run StudentDriver
Student Name: Student age: 0 grade: 0 My High School
Student Name: Joe Johnson age: 17 grade: 11 My High School
Student Name: Lucy Lee age: 16 grade: 10 My High School
>
```

Student object created from default constructor

Printing information from s1 using toString()

Using the methods to setName setAge and setGrade for s1.

Printing information again for s1

Student object created from overloaded constructor

Printing information from s2 using toString()

Creating a method to use instance variables & methods

Create ArrayList of students and add it to the field data

```
//Field Data.  Every object created gets a copy.  They are created private to protect the data

private String name;
private int age;
private int grade;
private final static String school = "My High School";

private ArrayList<Student> stu;
```

It is initialized in the constructors

```
public Student()
{
 name = "Student";
 age = 0;
 grade = 0;
 stu = new ArrayList<Student>();
}

public Student(String n, int a, int g)
{
 name = n;
 age = a;
 grade = g;
 stu = new ArrayList<Student>();
 }
```

Creating methods to use with ArrayList

The AddStudents method will add a student to the arraylist called stu in the field area.

```
//Field Data.  Every object created gets a copy.  They are created private to protect the data

private String name;
private int age;
private int grade;
private final static String school = "My High School";

private ArrayList<Student> stu;
```

```
public void addStudents(Student students)
{
stu.add(students);

}

public ArrayList<Student> getStudents()
{
 return stu;
}


public void checkAge(Student student)
{
 if(student.getAge() == 18)
 {
  System.out.print( student.getName() + "is 18");
 }
}
```

The getStudents method will return the ArrayList of students.

The checkAge(Student student)
Will check the age of the student. It calls the getAge method. If the student is 18 it will print out the name of the student by calling the getName() method.

In the main

The student objects created s1, s2, s3 are added to the ArrayList stu in the class by called the addStudents(Student s) method.

You loop through the ArrayList by using the getStudents method. It returns the ArrayList so you use it as an ArrayList. getStudent().size() will return the size of the ArrayList from the method getStudents.

You will print the arraylist by calling the getSTudents().get(i).getName() This returns just the name of the objects in the ArrayList.

Loop through the arraylist and call the checkAge method. The parameter is passed a student object from the arraylist

```
Student s3 = new Student("James John", 18, 12);

s1.addStudents(s1);
s1.addStudents(s2);
s1.addStudents(s3);


for(int i = 0; i < s1.getStudents().size(); i++)
{
 System.out.println(s1.getStudents().get(i).getName());

}

System.out.println("\n\n");

for(int i = 0; i < s1.getStudents().size(); i++)
{
s1.checkAge(s1.getStudents().get(i));

}

}
```

```
Joe Johnson
Lucy Lee
James John


James John is 18>
```