**UNDERSTANDING PACKAGES**

**AGENDA**

**Packaging Up Your Classes**

**Standard Packages**Standard Classes Encapsulating the Primitive Data Types  
Converting between Primitive Type Values and Strings

Converting Objects to Values

Autoboxing Values of Primitive Types

**Controlling Access To Class Members**

Using Access Attributes

Specifying Access Attributes

**Nested Classes**

**Packaging Up Your Classes**

* are collection of classes
* avoid possible name clashes
* class names in a package are all qualified by the package name.

Example: java.lang.String

* Every class in Java is contained in a package

**PACKAGES**

*Putting one of your classes in a named package …..*

package Geometry;

*accessible outside the package*

**public** class Line {

// Details of the class definition

}

***Example:*** *save all the files for the classes in the package “graphics”*

//*in the Draggable.java file*

**package graphics;**

public interface Draggable {

*A graphics interface and classes are in a package called graphics,*

. . .

}

//*in the Graphic.java file*

**package graphics;**

public abstract class Graphic {

. . .

}

//*in the Circle.java file*

**package graphics;**

public class Circle extends Graphic

implements Draggable {

. . .

}

//*in the Rectangle.java file*

**package graphics;**

public class Rectangle extends Graphic

implements Draggable {

. . .

}

//*in the Point.java file***package graphics;**

public class Point extends Graphic

implements Draggable {

. . .

}

//*in the Line.java file*

**package graphics;**

public class Line extends Graphic

implements Draggable {

. . .

}

**Packages and the Directory Structure**

*You can specify a package name as any sequence of names separated by periods.*

*collections of classes dealing with geometry….*

package Geometry.Shapes3D;

package Geometry.Shapes2D;

**Adding Classes from a Package to Your Program**

*Assuming the classes have been defined with the public keyword…..*

import Geometry.Shapes3D.**\***; // selects all the classes in the package,

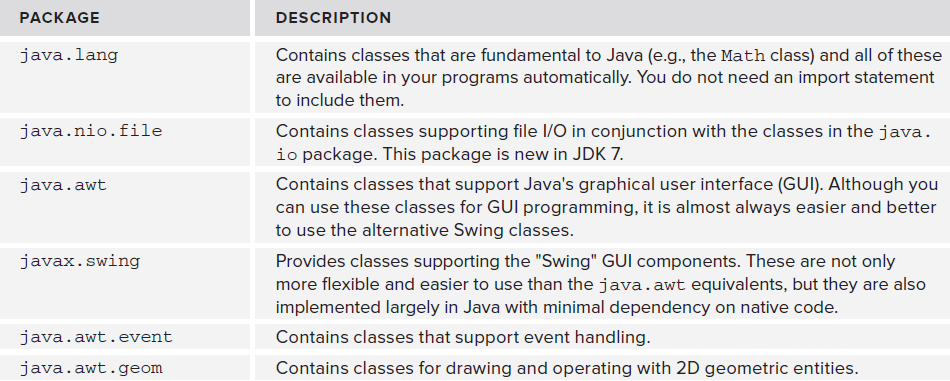
// you can refer to any public class in the package just by using the class name.

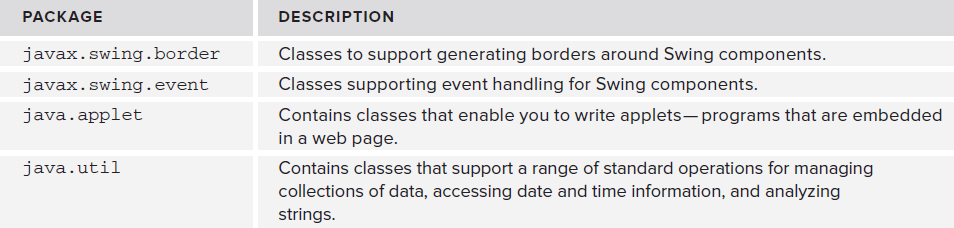
*It's usually better to import just the class names from a package that your code references.*

import Geometry.Shapes3D.Sphere; // Include the class Sphere

**Standard Packages**

*standard classes that are provided with Java are stored in standard packages*





**Standard Classes Encapsulating the Primitive Data Types**

*classes enable you to define objects that encapsulate values of each of the primitive data types in Java.*

Primitive Types Wrapper Classes

boolean java.lang.Boolean   
 byte  java.lang.Byte   
 char java.lang.Character   
 double java.lang.Double   
 float java.lang.Float   
 int java.lang.Integer   
 long java.lang.Long   
 short    java.lang.Short

* encapsulate a value of the corresponding primitive type
* provide methods for manipulating and interrogating objects of the class

**Converting between Primitive Type Values and Strings**

*Each class provides a static toString() method to convert a value of the corresponding primitive type to a String object.*

java.lang.**Integer**  
public static [String](https://docs.oracle.com/javase/7/docs/api/java/lang/String.html) toString(int i)

*There is also a non-static toString() method in each class that returns a String representation of a class object.*

java.lang.**Integer**

public [String](https://docs.oracle.com/javase/7/docs/api/java/lang/String.html) toString()

*There are methods to convert from a String object to a primitive type….*   
 java.lang.**Integer**  
public static int parseInt([String](https://docs.oracle.com/javase/7/docs/api/java/lang/String.html" \o "class in java.lang) s)

throws [NumberFormatException](https://docs.oracle.com/javase/7/docs/api/java/lang/NumberFormatException.html)

*To parse strings and return a value of the corresponding primitive type..*

Use methods parseShort(), parseByte(), parseInt(), and parseLong() in the classes for integer types

parseFloat() and parseDouble() for fl oating-point classes

parseBoolean() for Boolean.

java.lang.**Float**

public static float parseFloat([String](https://docs.oracle.com/javase/7/docs/api/java/lang/String.html) s)

throws [NumberFormatException](https://docs.oracle.com/javase/7/docs/api/java/lang/NumberFormatException.html)

*converts a string to an object of the class type containing the value represented by the string.*

java.lang.**Float**

public static [Float](https://docs.oracle.com/javase/7/docs/api/java/lang/Float.html) valueOf([String](https://docs.oracle.com/javase/7/docs/api/java/lang/String.html) s)

throws [NumberFormatException](https://docs.oracle.com/javase/7/docs/api/java/lang/NumberFormatException.html)

**Converting Objects to Values**

*Each class encapsulating a primitive data value also defines a xxxValue() method that returns the value that is encapsulated by an object as a value of the corresponding primitive type.*

java.lang.**Integer**

public byte byteValue()

public double doubleValue()

public float floatValue()

public int intValue()

public long longValue()

public short shortValue()

**Autoboxing Values of Primitive Types**

*Conversions from a primitive type to the corresponding class type are called boxing conversions/ autoboxing*

*compiler also inserts unboxing conversions to convert a reference to an object of a wrapper class for a primitive type*

public class AutoboxingInAction {

public static void main(String[] args) {

int[] values = { 3, 97, 55, 22, 12345 };

// Array to store Integer objects

Integer[] objs = new Integer[values.length];

// Call method to cause boxing conversions

for(int i = 0 ; i < values.length ; ++i) {

objs[i] = boxInteger(values[i]);

}

// Use method to cause unboxing conversions

for(Integer intObject : objs) {

unboxInteger(intObject);

}

}

// Method to cause boxing conversion

public static Integer boxInteger(Integer obj) {

return obj;

}

// Method to cause unboxing conversion

public static void unboxInteger(int n) {

System.out.println(“value = “ + n);

}

*Autoboxing is particularly useful when you need to insert values of primitive types into a collection*

**Controlling Access To Class Members**

How can you control the accessibility of class members from outside the class ?

* depends on what *access attributes* you have specified for the members of a class
* whether the classes are in the same package
* whether you have declared the class as public.

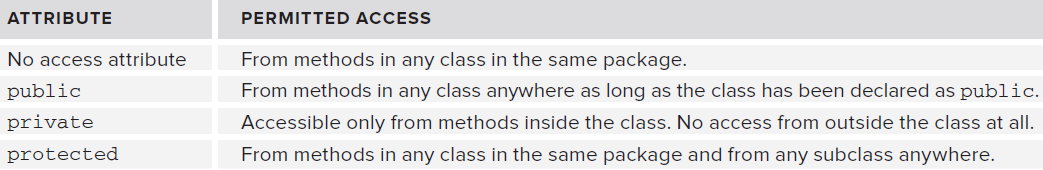
**Using Access Attributes**

*considering classes that are in the same package.*

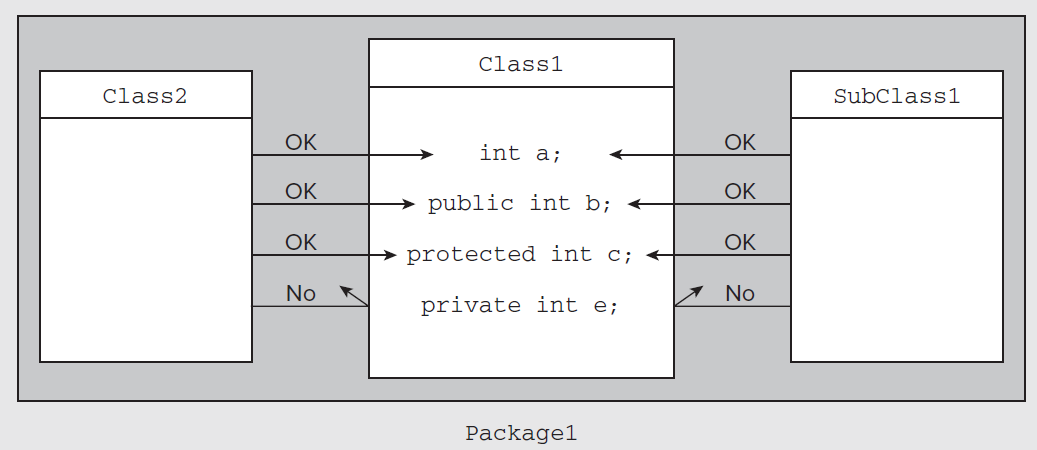
*any class has direct access to any other class name in the same package.*

*a public class in one package can be accessed from a class in another package*

*four possibilities when specifying an access attribute for a class member*



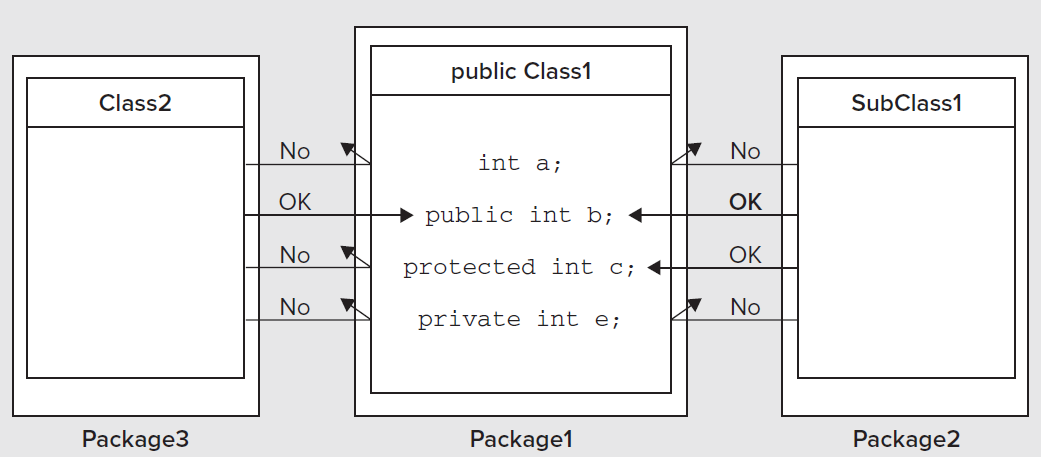
The following figure shows the access allowed between classes within the same package.



*private members of the class - can be accessed only by methods in the same class.*

*class definition must have an access attribute of public if it is to be accessible from outside the package .*

In the following figure, classes seeking access to the members of a public class are in different packages.



*Here access is more restricted.*

*Only public members of Class1 that can be accessed from an ordinary class, Class2, in another package,*

*From a subclass of Class1 that is in another package, the members of Class1 without an access attribute cannot be reached, and neither can the private members*

**Specifying Access Attributes**



*Here is the Point class with access attributes defined for its members:*

**import static java.lang.Math.sqrt;**

**public class Point {**

// Create a point from its coordinates

**public Point(double xVal, double yVal) {**

x = xVal;

y = yVal;

}

// Create a Point from an existing Point object

**public Point(fi nal Point aPoint) {**

x = aPoint.x;

y = aPoint.y;

}

// Move a point

**public void move(double xDelta, double yDelta) {**

// Parameter values are increments to the current coordinates

x += xDelta;

y += yDelta;

}

// Calculate the distance to another point

**public double distance(fi nal Point aPoint) {**

**return sqrt((x - aPoint.x)\*(x - aPoint.x)+(y - aPoint.y)\*(y - aPoint.y));**

}

// Convert a point to a string

**public String toString() {**

return Double.toString(x) + “, “ + y; // As “x, y”

}

// Coordinates of the point

**private double x;**

**private double y;**

}

*Instance variables x and y can be set or modified through methods within the class, either with constructors or the move() method.*

*To obtain the values of x and y from outside the class…*

public double getX() {

*accessor* methods

return x;

}

public void setX(double inputX) {

*mutator* methods

x = inputX;

}

*Note: It may seem odd to use a method to alter the value of a private data member when you could*

*advantage of using a method in this way is that you can apply validity checks on the new value*

*don't include a mutator method, if you don't want to allow the value of a private member to be changed*

**Nested Classes**

public class Outside {

member of the enclosing class

// Nested class

**public** class Inside {

// Details of Inside class...

}

// More members of Outside class...

}

**Using a Non-Static Nested Class**

It has access to all of its enclosing class' members including private

**OuterClass** outerObject = new **OuterClass**();  **OuterClass.InnerClass**  innerObject = outerObject**.**new **InnerClass**();

*creating an object of type InnerClass in the context of the object OuterClass.*

*Note:* nested class here has meaning only in the context of an object of type OuterClass….

*You could create a new InnerClass object from within a method of the object OuterClass:*

InnerClass inner = new InnerClass(); // Define a nested class object

This statement is equivalent to:

this.InnerClass inner = this.new InnerClass(); // Define a nested class object

**Static Nested Classes**

*To make objects of a nested class type independent of objects of the enclosing class type….*

**OuterClass.StaticNestedClass**

For example, to create an object for the static nested class, use this syntax:

**OuterClass.StaticNestedClass** nestedObject = new **OuterClass.StaticNestedClass**();

**Local Classes:**

**public void doStuff() {**

* are like inner (non-static nested) classes
* can only be accessed from inside the method

**class MyInner {  
 public void seeOuter() {  
 System.out.println("x is " + x);  
 } }**

**MyInner i = new MyInner();**

**i.seeOuter();  
 }**

**Anonymous Classes:**

are classed without a class name

are declared as either subclasses of an existing class, or as implementations of some [interface](http://tutorials.jenkov.com/java/interfaces.html).

are defined when they are instantiated

class Ferrari {  
public void drive() {  
System.out.println("Ferrari");  
 }  
 }  
class Car {  
 Ferrari p = new Ferrari() {

Ferrari reference variable (p) refers to an instance of an anonymous (unnamed) subclass of Ferrari

public void drive() {   
 System.out.println("anonymous Ferrari");  
 }  
 };  
}