Aylin Göke

Arda Mızrak

Cem Şengel

Ergül Dilan Çelebi

WEEK 8 LECTURE NOTES
April 1-3, 2008
ARRAYLIST CLASS

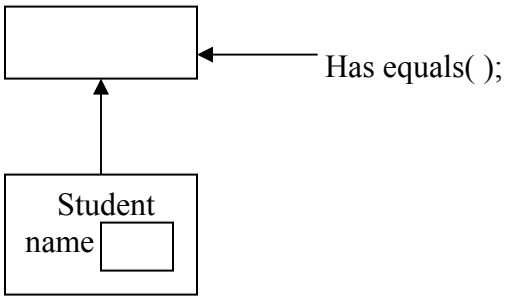
ArrayList class is a part of java.util library. It is a kind of array that contains objects. It is called just like that;

```
Object [ ] a = new Object [10];
 ArrayList books = new ArrayList( );
```

The Methods of ArrayList Class

add (index:int, o:object):void
add a new element o at the specified index in the list.
clear ( ): void
removes all elements of an ArrayList.
contains (o:object) : Boolean
it returns true if the object is an element of a list.
get (index:int):object
gives the object of the specified element of the list.
indexOf (o:object) : int
returns the first occurrence.
isEmpty ( ) : Boolean
returns true if the list contains null elements.
lastIndexOf (o:object) : int
returns the last occurrence.
remove (o:object) : boolean
size ( ) : int

How to Implement Equals

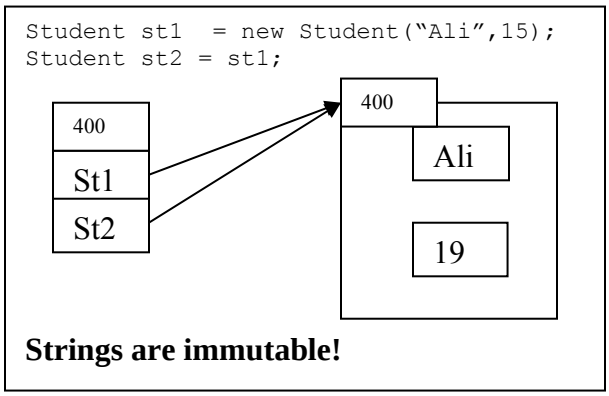


1.
```
public boolean equals( Student otherStudent) {
      return (name.equals(otherStudent.name));
}
```
Consider equals( ) method of Object Class;

```
public boolean equals(Object otherObject)
```
implementation no.1 : overloads the equals( );
method defined in the Object Class.

2. Some library methods assume the following implementation for the equals( ) method :
The following <u>overrides</u> the equals( ) method of the Object Class

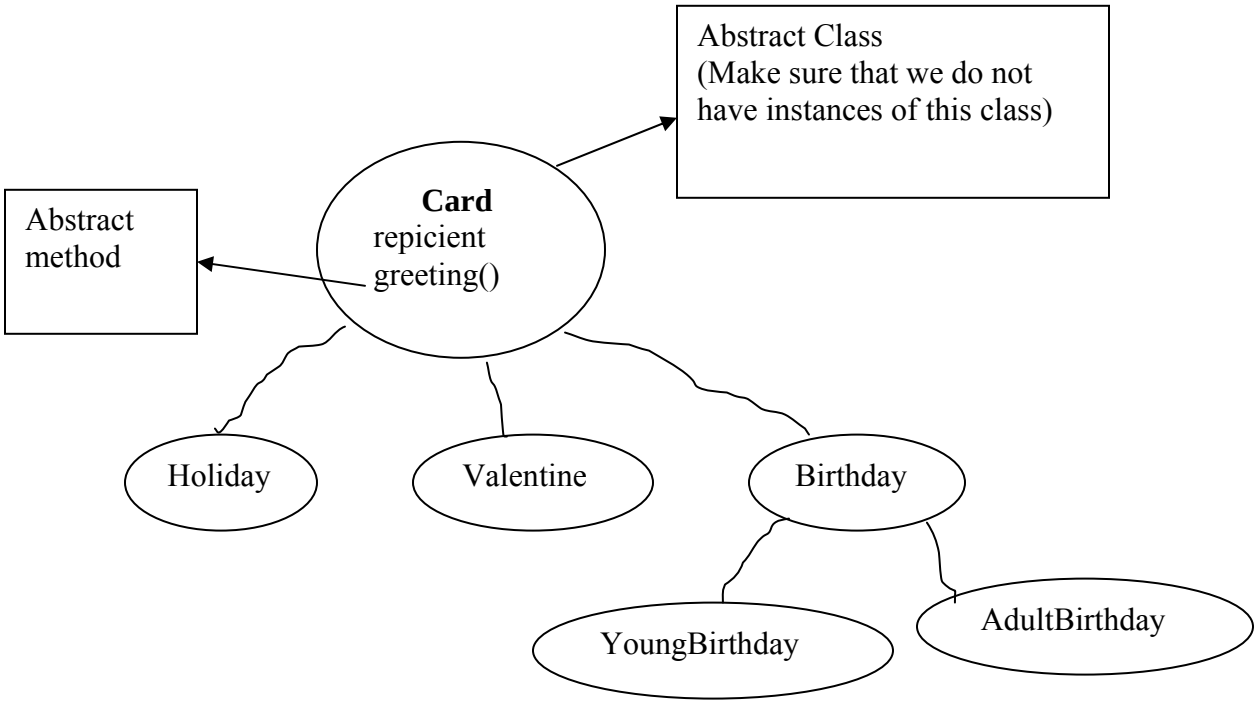
```
public boolean equals(Object otherObject) {
      Student otherStudent = (Student) otherObject;
      return( name.equals(otherStudent.name));
}
 Student st1 = Student("Ali",15);
Student st2;
if  (st1.equals(st2))
      system.out.println("the same student");
```



Strings are immutable!

3.
```
public boolean equals(Object otherObject) {
      if(otherObject = = null)
            return false;
      else if (this.getClass( ) ! = otherObject.getClass( ))
            return false;
      else {
            Student otherStudent = (Student) otherObject;
            Return (name.equals(otherStudent.name));
      }
}
```

ABSTRACT CLASS



package AbstractClassNotes;

```
public abstract class Card
{ String recipient;
  public abstract void greeting(); }
```

***it is possible to have non-abstract methods in an abstract class. The logic of using abstract method is to obtain different implementations for subclasses.**
***By this reason, abstract methods have no implementation in the abstract class.**

```
package AbstractClassNotes;
public class Holiday extends Card
   { public Holiday(String r)
     { recipient=r;} //constructor
   public void greeting()
    {System.out.println("Dear " + recipient + ", Season's
Greetings!");}
      //the implementation of the greeting() method for Holiday class
}

package AbstractClassNotes;
public class Valentine extends Card
{ int kisses;
```

```
 public Valentine(String r, int k)
   {recipient=r; kisses=k;} //constructor
 public void greeting()
   {System.out.println("Dear " + recipient + ", Love and kisses");
    for (int i=0; i<kisses; i++)
    System.out.println("*");
//the implementation of the greeting() method for Valentine class
 }
}

package AbstractClassNotes;
public class Birthday extends Card
{ int age;
   public Birthday(String r, int years)
     {recipient=r; age=years; } //constructor
  public void greeting()
    {System.out.println("Dear " + recipient + ", Happy " + age + " th
Birthday"); }
    //the implementation of the greeting() method for Birthday class
    }

package AbstractClassNotes;
public class YouthBirthday extends Birthday
{ public YouthBirthday(String r, int years)
    {super(r, years);} //constructor using Birthday super class
  public void greeting()
    {super.greeting();
     System.out.println("How you have grown!");
     //this methods overrides the greeting() method found in Birthday
class
    }
}

package AbstractClassNotes;
public class AdultBirthday extends Birthday
{ public AdultBirthday(String r, int years)
      {super(r, years);} //constructor using Birthday super class
 public void greeting()
     {super.greeting();
     System.out.println("You haven't changed at all!!!");
   //this methods overrides the greeting() method found in Birthday
class
    }
}
```

THE TESTER:

```
package AbstractClassNotes;
import java.util.ArrayList;
public class CardTester
{ public static void main (String[] args)
  { //Card mycard=new Card();
     //mycard.greeting();
     //this gives compilation error as Card is an abstract class;
     //abstract class does not contain a constructor;
```

```
String me;
me="Peter";
Holiday hol=new Holiday(me);
hol.greeting();

Birthday bd=new Birthday(me, 21);
bd.greeting();

Valentine val=new Valentine(me, 7);
val.greeting();
```

Output:
Dear Peter, Season's Greetings!
Dear Peter, Happy 21 th Birthday
Dear Peter, Love and kisses
*
*
*
*
*
*

```
//We can also use Card as a type of the instances of  subclasses.
      Card yourcard=new Valentine("Bob", 3);
      yourcard.greeting();
 // Birthday or Card can be types of the instances of Youth or Adult
Birthday classes
      YouthBirthday yb=new YouthBirthday("Doğa", 2);
      yb.greeting();
      Birthday newyb=new YouthBirthday("Doğa", 2);
      newyb.greeting();
       System.out.println();
      AdultBirthday ab=new AdultBirthday("Nur", 58);
      ab.greeting();
      Card newab=new AdultBirthday("Nur", 58);
      newab.greeting();
```

Output:

Dear Bob, Love and kisses
*
*
*
Dear Doğa, Happy 2 th Birthday
How you have grown!
Dear Doğa, Happy 2 th Birthday
How you have grown!

Dear Nur, Happy 58 th Birthday
You haven't changed at all!!!
Dear Nur, Happy 58 th Birthday
You haven't changed at all!!!

/* Ass you see, result does not change when we use Card or Birthday instead of YoungBirthday or AdultBirthday as they are the subclasses of Card and Birthday.

We know that we cannot create the instances of Card as it is an abstract class but we can create an Card array and assign the objects of subclasses to this array. */

```
Card [] array=new Card [2];
   array[0]=new Birthday("Pelin", 20);
   array[1]=new Valentine("mm", 2);
   array[0].greeting();
   array[1].greeting();
```

Output:
Dear Pelin, Happy 20 th Birthday
Dear mm, Love and kisses
*
*

//We can also create an arraylist and assign all the instances of subclasses into this arraylist. "hol, bd, val…" the objects that we have created previously.

```
    ArrayList myarray= new ArrayList();
       myarray.add(hol);
       myarray.add(bd);
       myarray.add(val);
       myarray.add(yourcard);
    myarray.add(yb);
       myarray.add(ab);
```

// As our general super class is Card, we can assign Card type for each subclass and its execution is specific to subclass type

```
for(int j=0; j<myarray.size(); j++)
      {Card samplecard= (Card) myarray.get(j);
      samplecard.greeting();}
  }
}
```

Output:
Dear Peter, Season's Greetings!
Dear Peter, Happy 21 th Birthday
Dear Peter, Love and kisses
*
*
*
*
*
*
*
Dear Bob, Love and kisses
*
*
*
Dear Doğa, Happy 2 th Birthday
How you have grown!
Dear Nur, Happy 58 th Birthday
You haven't changed at all!!!

Questions for Abstract Class concept:

Indicate true or false for the following statements: (questions are taken from review questions of our text book.)
1. An abstract class can have instances created using the constructor of the abstract class.
Answer: No, but an instance of its concrete subclass is also an instance of the parent abstract class. You cannot create an object using the constructor of the abstract class.

However, you can create an object using a concrete subclass of the abstract class. This object is an instance of the subclass and it is also an instance of the abstract class. Like our Card example, you can assign: Card mycard=new Holiday("murat");
2. An abstract class can be extended.
Answer: True as we extends it with different subclasses such as Holiday, Birthday…
3. A subclass of a non-abstract superclass cannot be abstract.
Answer: False as it is possible to have an abstract subclass.
4. An abstract method must be non-static
Answer: True as methods are implemented differently depending on the subclasses.

MULTIPLE INHERITENCE

// We can't have two super class for a class in java environment. However we have interfaces so that we can do a similar task.

Interfaces:

They're similar to abstract classes. Interfaces make sure that certain methods are implemented in classes that use an interface which contains the method header lines.

Example:

// Represents the interface for an object that can be assigned an explicit complexity.

```
public interface Complexity{
      public void setComplexity(int complexity);
      public int getComplexity();
}
```

// Here, note that we can only have constant definitions in interfaces.

//Represents a question and its answer

```
public class Question implements Complexity{

      // The general structure is; public class className1 extends
className2 implements Interface1, Interface2..

      private String question, answer;
      private int complexityLevel;
      public Question(String query, String result){
            question = query;
            answer = result;
            complexityLevel = 1;
      }
      public void setComplexity(int level){
            complexityLevel=level;
      }
      public int getComplexity() {
            return complexityLevel;
      }
      public String getQuestion(){
            return question;
```

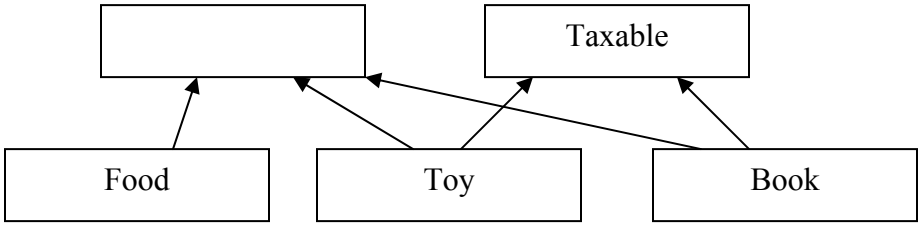
```
	}
	public String getAnswer(){
		return answer;
	}
	public boolean answerCorrect(String candidateAnswer){
		return(answer.equalsIgnoreCase(candidateAnswer));
	}
	public String toString(){
		String result = ("The Question is : " + question  + ", and
the answer is : " + answer);

}
```

TESTER:

```
public class MiniQuiz{
	public static void main(String [] args){
		Question q1,q2;
		q1=new Question("What is the capital of Jamaica?" ,
"Kingston");
		q1.setComplexity(7);
		q2=new Question("What is your favorite class?", "Calculus" );
		q2.setComplexity(1);
		System.out.println( "question1 :" + q1);
		System.out.println( "question2 :" + q2);
	}//end of main
}//end of class MiniQuiz
```

We can summarize the topic with a diagram followed below;



// In this case, Food , Toy , and Book are subclasses of class Good, And only Toy and Book are taxable.

```
public class Goods{
	String description;
	double price;
	public Goods(String desc, double pr){
		description = desc;
		price = pr;
	}
```

```
	public String toString(){
		return ("item :" + description + "price :" + price);
	}
}//end of class Goods

// Interface Taxable

public interface Taxable{
	public final double TAX_RATE=0.18;
	public double calculateTax();
}
public class Food extends Goods{
	private double calories;
	public Food(String desc, double pr, double cal){
		super(desc, pr);
		calories = cal;
	}
	public String toString(){
		return (super.toString() + "calories :" + calories);
	}
}
public class Toy extends Goods implements Taxable{
	private int minimumAge;
	public Toy(String desc, double pr, int minage){
		super(desc, pr);
		minimumAge=minage;
	}
	public String toString(){
		return (super.toString() + "minimum age :" + minimumAge);
	}
	public double calculateTax(){
		return(price*TAX_RATE);
}
```

```
public class Book extends Goods implements Taxable{
	private String author;
	public Toy(String desc, double pr, String author){
		super(desc, pr);
		this.author= author;
	}
	public String toString(){
		return (super.toString() + "Author : " + author);
	}
	public double calculateTax(){
		return(price*TAX_RATE);
}
```

Questions to think about multiple inheritance:

1) May we use any kind of definition in interfaces?
1a) No, the one must only use constant definitions in interfaces, thus they're implemented in the class.
2) May we implement more than one interface from one class?
2a) Yes, there is no limit.
3) In what sense interfaces are useful?
3a) By using interfaces, we make sure that we implement the methods that are needed.