**Assignment 5 Turn in Sheet Name:**

Lab Questions (**Total 50 Pts.**)

Big Java, Late Objects / Java for Everyone, 2e

**Chapter Number: 9 Inheritance and Interfaces**

1.) *This continues Assignment 3 (after 3.10):*

1.1) (5 pts) Implement toString methods for the Card class and its three subclasses. The methods should print:

the name of the class

the values of all instance variables (including inherited instance variables)

Typical formats are:

Card[name=Edsger W. Dijkstra]

CallingCard[name=Bjarne Stroustrup][number=4156646425,pin=2234]

Write the code for your toString methods.

|  |
| --- |
| **//CARD CLASS**  @Override  public String toString()  {  return "Card [name = " + name + "]";  }  **// ID CARD CLASS**  @Override  public String toString()  {  return "ID Card\t\t[name = " + super.getName() + "] [ID Number = "  + idNumber + "]";  }  **// CALLING CARD CLASS**  @Override  public String toString()  {  return "Calling Card\t[name = " + super.getName() + "] [number = "  + cardNumber +", pin = "+ pinNumber +"]";  }  **// DRIVER LICENSE CLASS**  @Override  public String toString()  {  return "Driver License\t[name = " + super.getName() +  "] [Expiration Year = " + expirationYear +"]";  }  public static void main(String [] args)  {  CallingCard cc\_test = new CallingCard("Tyler Quayle", "1230", "1988");  DriverLicense dl\_test = new DriverLicense("Tyler Quayle", 2016);  IDCard id\_test = new IDCard("Tyler Quayle", "12421");    System.out.println(cc\_test);  System.out.println(dl\_test);  System.out.println(id\_test);  }  **OUTPUT:**  **Calling Card [name = Tyler Quayle] [number = 1230, pin = 1988]**  **Driver License [name = Tyler Quayle] [Expiration Year = 2016]**  **ID Card [name = Tyler Quayle] [ID Number = 12421]** |

1.12) (5 pts) Implement equals methods for the Card class and its three subclasses. Cards are the same if the objects belong to the same class, and if the names and other information (such as the expiration year for driver licenses) match.

Give the code for your equals methods.

|  |
| --- |
| **// CARD**  public boolean equals(Card rightCard)  {  if(name.equals(rightCard.getName()))  return true;  else  return false;  }  **// CALLING CARD**  public boolean equals(CallingCard rightCard)  {  if(super.equals(rightCard))  if(cardNumber.equals(rightCard.getCardNumber()))  if(pinNumber.equals(rightCard.getpinNumber()))  return true;    return false;  }  **// ID CARD**  public boolean equals(IDCard rightCard)  {  if(super.equals(rightCard))  if(idNumber.equals(rightCard.getIDNumber()))  return true;  return false;  }  **// DRIVERS LICENSE**  public boolean equals(DriverLicense rightCard)  {  if(super.equals(rightCard))  if(expirationYear == rightCard.getExpirationYear())  return true;    return false;  }  **// MAIN**  public static void main(String [] args)  {  CallingCard cc\_test = new CallingCard("Tyler Quayle", "1230", "1988");  CallingCard cc\_equal\_test = new CallingCard("Tyler Quayle", "1230", "1922");    DriverLicense dl\_test = new DriverLicense("Tyler Quayle", 2016);  DriverLicense dl\_equal\_test = new DriverLicense("Tyler Quayle", 2016);    IDCard id\_test = new IDCard("Tyler Quayle", "12421");  IDCard id\_equal\_test = new IDCard("Tyler Quayle", "999999");    System.out.println("Testing Calling Card Equals Method: Expecting DNE");  if(cc\_test.equals(cc\_equal\_test))  System.out.println(cc\_test + "\n\t Equals \n" + cc\_equal\_test);  else  System.out.println(cc\_test + "\n\tDoes not Equal\n" + cc\_equal\_test);    System.out.println("\nTesting Driver License Equals Method: Expecting EQUAL");  if(dl\_test.equals(dl\_equal\_test))  System.out.println(dl\_test + "\n\t Equals\n" + dl\_equal\_test);  else  System.out.println(dl\_test + "\n\t Does not Equal\n" + dl\_equal\_test);    System.out.println("\nTesting Driver License Equals Method: Expecting DNE");  if(id\_test.equals(id\_equal\_test))  System.out.println(id\_test + "\n\t Equals\n" + id\_equal\_test);  else  System.out.println(id\_test + "\n\t Does not Equal \n" + id\_equal\_test);  }  **OUTPUT:**  **Testing Calling Card Equals Method: Expecting DNE**  **Calling Card [name = Tyler Quayle] [number = 1230, pin = 1988]**  **Does not Equal**  **Calling Card [name = Tyler Quayle] [number = 1230, pin = 1922]**  **Testing Driver License Equals Method: Expecting EQUAL**  **Driver License [name = Tyler Quayle] [Expiration Year = 2016]**  **Equals**  **Driver License [name = Tyler Quayle] [Expiration Year = 2016]**  **Testing Driver License Equals Method: Expecting DNE**  **ID Card [name = Tyler Quayle] [ID Number = 12421]**  **Does not Equal**  **ID Card [name = Tyler Quayle] [ID Number = 999999]** |

1.13) (5 pts) Change the Card class and give protected access to name. Would that change simplify the toString method of the CallingCard class? How?

Is this change advisable?

**Yes, it would change the simplicity of the toString methods as you can now just call *super.variable* and can forgo any get() methods. However in java, as with any programming language you want to make your class as private as possible so that subclasses may not simply change the variable itself as a normal set() method may also do other calculations.**

2) (10 pts) Start by building a default directory that contains the class files AClass, BClass, and ASubClass. All the classes you need for this assignment are already written and can be copied from the listings below. The default directory should contain a subdirectory called Temp that contains the class file CClass.

Inside AClass and ASubClass there is a method called addem which contains a number of lines that are commented out. Uncomment one line at a time. Write a comment under each line that describes why the line above it will or will not compile. Be sure to keep the import and package statements in the original code. **Comment out any lines that won’t compile in the final submission.**

If you forget to code an access modifier on a class variable (that is, you don’t code public, protected, or private), Java assigns the variable “package access” which opens it up to modification from any class in the same package as the class that contains the unmodified variable. Most programmers consider this a bad idea as it breaks the object-oriented principle of data encapsulation. This insidious effect occurs when you are forgetful and leave off the access modifier accidentally. For that reason, you need to be aware of this default behavior in case you are careless.

**Explain** how package access variables differ from protected access variables.

|  |  |  |  |  |
| --- | --- | --- | --- | --- |
| **Access Levels** | | | | |
| **Modifier** | **Class** | **Package** | **Subclass** | **World** |
| public | Y | Y | Y | Y |
| protected | Y | Y | Y | N |
| *no modifier* | Y | Y | N | N |
| private | Y | N | N | N |

**As you can see, a default or ‘package’ variable cannot be accessed by subclasses, which could lead to numerous errors if it does not become *protected* or *private*. So it is really a private/public hybrid. As you do not need to have get/set methods like a private variable and it is only available to things within the package like a private.**

import Temp.\*;

public class AClass

{

private int aprivate;

protected int aprotected;

public int apublic;

int apackage;

static int noAObjects = 0;

/\*\*

Think about these declarations

\*/

BClass bobj = new BClass();

CClass cobj = new CClass();

/\*\*

Constructor for objects of class AClass.

\*/

public AClass()

{

// Initialize instance variables

aprivate = 1;

aprotected = 2;

apublic = 3;

apackage = 4;

noAObjects ++;

}

public int addem()

{

**//System.out.println(bobj.bprivate);**

// Cannot access, private variable

**System.out.println(bobj.bprotected);**

// Can access, protected can be accessed within package

**System.out.println(bobj.bpublic);**

// Public variable, can access

**System.out.println(bobj.bpackage);**

// Can Access public and in same package

**//System.out.println(cobj.cprivate);**

// Private, cannot access

**//System.out.println(cobj.cprotected);**

// Protected in different package. cannot access

**System.out.println(cobj.cpublic);**

// Public cannot access

**//System.out.println(cobj.cpackage);**

// Different package, cannot access

return aprivate + aprotected + apublic + apackage;

}

}

---------------------

public class BClass

{

private int bprivate;

protected int bprotected;

public int bpublic;

int bpackage;

/\*\*

Constructor for objects of class BClass.

\*/

public BClass()

{

// Initialize instance variables

bprivate = 1;

bprotected = 2;

bpublic = 3;

bpackage = 4;

}

public int addem()

{

return bprivate + bprotected + bpublic + bpackage;

}

}

---------------------

import Temp.\*;

public class ASubClass extends AClass

{

private int adprivate;

protected int asprotected;

public int aspublic;

int aspackage;

/\*\*

Think about these declarations

\*/

BClass bobj = new BClass();

CClass cobj = new CClass();

/\*\*

Constructor for objects of class ASubClass

\*/

public ASubClass()

{

// Initialize instance variables

asprivate = 1;

asprotected = 2;

aspublic = 3;

aspackage = 4;

}

public int addem()

{

**// System.out.println(bobj.bprivate);**

//Cannot access Private Directly

**System.out.println(bobj.bprotected);**

// Protected, can be accessed by any subclasses without need for get methods

**System.out.println(bobj.bpublic);**

// Public variable, can be accessed by anything at anytime

**System.out.println(bobj.bpackage);**

// Package variable, can be accessed by any class in the package, but no

**//System.out.println(cobj.cprivate);**

//Cannot access Private Directly

**//System.out.println(cobj.cprotected);**

// Cannot be accessed, as ASub does not extend CClass so protected are not accessable

**System.out.println(cobj.cpublic);**

// Public variable, can be accessed by anything at anytime

**//System.out.println(cobj.cpackage);**

// Cpackage cannot be accessed outside of CClass

**// System.out.println(aprivate);**

// Private variable, cannot be accessed

**System.out.println(aprotected);**

// Protected, can be accessed by ASub, since it is subclass

**System.out.println(apublic);**

// Public, can be accessed at anytime

**System.out.println(apackage);**

// ASub is subclass of A, can access AClass

return adprivate + asprotected + aspublic + aspackage

+ aprotected + apublic + apackage;

}

}

---------------------

package Temp;

public class CClass

{

private int cprivate;

protected int cprotected;

public int cpublic;

int cpackage;

/\*\*

Constructor for objects of class CClass

\*/

public CClass()

{

// Initialize instance variables

cprivate = 1;

cprotected = 2;

cpublic = 3;

cpackage = 4;

}

public int addem()

{

return cprivate + cprotected + cpublic + cpackage;

}

}

3) (15 pts) It is often the case that two or more classes share a common set of methods. For programming purposes we might wish to treat the objects of those classes in a similar way by invoking some of their common routines.

For example, the Dog and Cat classes listed below agree on the void method speak. Because Dog and Cat objects have the ability to “speak,” it is natural to think of putting both types of objects in an ArrayList and invoking speak on every object in the list. Is this possible? Certainly we could create an ArrayList of Dog that would hold all the Dog objects, but can we then add a Cat object to an ArrayList of Dog?

Try running the main program below as it is written. Run it a second time after uncommenting the line that instantiates a Cat object and tries to add it to the ArrayList.

import java.util.\*;

public class AnimalRunner

{

public static void main(String[] args)

{

ArrayList<Dog> dogcatList = new ArrayList<Dog>();

dogcatList.add(new Dog("Fred"));

// dogcatList.add(new Cat("Wanda"));

}

}

-------------------

public class Dog

{

private String name;

public Dog(String name)

{

this.name = name;

}

public void speak()

{

System.out.println("Woof! Woof!");

}

public String toString()

{

return "Dog: " + name;

}

}

-------------------

public class Cat

{

private String name;

public Cat(String name)

{

this.name = name;

}

public void speak()

{

System.out.println("Meow! Meow!");

}

public String toString()

{

return "Cat: " + name;

}

}

Our experiment to add Cat objects to an ArrayList of Dog objects failed. Perhaps we should try using the original Java ArrayList without generics? Try running the code below as it is written along with the Dog and Cat classes defined above. Run it a second time after uncommenting the line that invokes speak.

import java.util.\*;

public class AnimalRunner

{

public static void main(String[] args)

{

ArrayList dogcatList = new ArrayList();

dogcatList.add(new Dog("Fred"));

// dogList.add(new Cat("Wanda"));

for (Object obj : dogcatList)

{

// obj.speak();

}

}

}

The experiment shows that we are now able to add Dog and Cat objects to the ArrayList, but there is a compile error on the line obj.speak because obj is an Object reference variable and the class Object doesn’t contain a speak method. We need a reference variable that can refer to Dog and Cat objects and which also allows us to invoke speak. The solution to the problem uses interfaces.

First create an interface called Speakable that specifies a void speak() method. Be sure to modify the Dog and Cat classes to indicate that they implement the Speakable interface. For example, in the case of the Dog class, we will code public class Dog implements Speakable. Be sure to make a similar change in the declaration of the Cat class.

The term Speakable can be used to create Speakable references. Using generics, create an ArrayList of Speakable objects in the main method. Modify the for loop so that it iterates over Speakable objects. Try adding the Dog and Cat objects and invoking the speak method on each object. Does this work?

**Yes, it does work as an interface array can be filled with any classes that implement the Speakable interface;**

|  |
| --- |
| **// in speakable.java**  interface Speakable {  public void speak();  }  **// in animalrunner.java**  public class AnimalRunner  {  public static void main(String[] args)  {  Speakable [] animalTalk = new Speakable[2];  animalTalk[0] = new Dog("Fred");  animalTalk[1] = new Cat("Wanda");  for (Speakable obj : animalTalk)  {  obj.speak();  }  }  }  **OUTPUT:**  **Woof! Woof!**  **Meow! Meow!** |