**Assignment 3 Turn in Sheet Name: Tyler Quayle**

Lab Questions (**Total 50 Pts. + 14 EXTRA Pts.**)

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

**Chapter Number: 3 Decisions**

1) (4 pts.) To compare strings in Java you can’t simply use the relational operators. You need to use methods of the String class: equals(), compareTo(), and substring(). Assume the following code:

String word1 = "catalog";

String word2 = "cat";

Write the following conditions in Java:

* 1. word1 is lexicographically greater than "aaa"

|  |
| --- |
| System.out.println(Word1.compareTo(“aaa”));  **OUTPUT: 2**  **// if greater than 0, “aaa” comes before word1.**  **// if 0, words are equal**  **// if less than 0, “aaa” comes after word1.** |

* 1. word1 is lexicographically equal to word2

|  |
| --- |
| System.out.println(word2.compareTo(word1.substring(0,3)));  **OUTPUT: 0**  **// If 0, words are equal, in this case only comparing first 3 letters // of word1 since doing word1.compareTo(word2) will give 4 as the**  **// answer** |

* 1. word1 is lexicographically less than word2

|  |
| --- |
| System.out.println(word2.compareTo(word1));  **OUTPUT: -4** |

* 1. word1 and word2 have the same three-letter prefix

|  |
| --- |
| System.out.println(word2.equals(word1.substring(0,3)));  **OUTPUT: True** |

**Chapter Number: 4 Loops**

2) (10 pts.) One of the oldest numerical algorithms was described by the Greek mathematician, Euclid, in 300 B.C. That algorithm is described in Book VII of Euclid’s multi-volume work *Elements*. It is a simple but very effective algorithm that computes the greatest common divisor of two given integers.

For instance, given integers 24 and 18, the greatest common divisor is 6, because 6 is the largest integer that divides evenly into both 24 and 18. We will denote the greatest common divisor of *x* and *y* as gcd(*x*, *y*). The algorithm is based on the clever idea that the gcd(*x*, *y*) = gcd(*x* – *y*, *y*) if *x* >= *y*. The algorithm consists of a series of steps (loop iterations) where the “larger” integer is replaced by the difference of the larger and smaller integer.

In the example below, we compute gcd(72, 54) and list each loop iteration computation on a separate line. The whole process stops when one of the integers becomes zero. When this happens, the greatest common divisor is the non-zero integer.

gcd(72, 54) = gcd(72 – 54, 54) = gcd(18, 54)

gcd(18, 54) = gcd(18, 54 – 18) = gcd(18, 36)

gcd(18, 36) = gcd(18, 36 – 18) = gcd(18, 18)

gcd(18, 18) = gcd(18 – 18, 18) = gcd(0, 18) = 18

To summarize:

Create a loop, and subtract the smaller integer from the larger one (if the integers are equal you may choose either one as the “larger”) during each iteration.

Replace the larger integer with the computed difference.

Continue looping until one of the integers becomes zero.

Print out the non-zero integer.

Use the code below to prompt the user for the two integers.

|  |
| --- |
| public static void main(String[] args)  {  Scanner in = new Scanner(System.in);  System.out.println("Enter the first integer: ");  int x = in.nextInt();  System.out.println("x = " + x);  System.out.println("Enter the second integer: ");  int y = in.nextInt();  System.out.println("y = " + y);  // Your gcd computation code goes here  while(x != 0)  {  if(x > y)x = x - y;  else if(x < y) y = y - x;  else if(x == y) x = y - x;  }  System.out.println("GCD: " + y);  }  **OUTPUT:**  **Enter the first integer:**  **72**  **x = 72**  **Enter the second integer:**  **54**  **y = 54**  **GCD: 18** |

**Chapter Number: 9 Inheritance and Interfaces**

(Each 3.x 5 pts.)

3.1) Consider using the following Card class.

public class Card

{

private String name;

public Card()

{

name = "";

}

public Card(String n)

{

name = n;

}

public String getName()

{

return name;

}

public boolean isExpired()

{

return false;

}

public String format()

{

return "Card holder: " + name;

}

}

Use this class as a superclass to implement a hierarchy of related classes:

|  |  |
| --- | --- |
| Class | Data |
| IDCard | ID number |
| CallingCard | Card number, PIN |
| DriverLicense | Expiration year |

Write declarations for each of the subclasses. For each subclass, supply private instance variables. Leave the bodies of the constructors and the format methods blank for now.

|  |
| --- |
| public static class IDCard extends Card  {  private String idNumber;    public IDCard(){}    public String format()  {  return null;  }  }  Public static class CallingCard extends Card  {  private String cardNumber;  private String pinNumber;    public CallingCard(){}    public String format(){return null;}  }  Public static class DriverLicense extends Card  {  private int expirationYear;    public DriverLicense(){}    public String format(){return null;}  } |

3.2) Implement constructors for each of the three subclasses. Each constructor should call the superclass constructor to set the name. Here is one example:

public IDCard(String n, String id)

{

super(n);

idNumber = id;

}

|  |
| --- |
| Public static class IDCard extends Card  {  private String idNumber;    public IDCard(String s, String id)  {  super(s);  idNumber = id;  }    public String format()  {  return null;  }  }  Public static static class CallingCard extends Card  {  private String cardNumber;  private String pinNumber;    public CallingCard(String s, String car, String pin)  {  super(s);  cardNumber = car;  pinNumber = pin;  }    public String format()  {  return null;  }  }  Public static class DriverLicense extends Card  {  private int expirationYear;    public DriverLicense(String s, int exp)  {  super(s);  expirationYear = exp;  }    public String format()  {  return null;  }  } |

3.3) Replace the implementation of the format method for the three subclasses. The methods should produce a formatted description of the card details. The subclass methods should call the superclass format method to get the formatted name of the cardholder.

|  |
| --- |
| public static class IDCard extends Card  {  private String idNumber;    public IDCard(String s, String id)  {  super(s);  idNumber = id;  }    public String format()  {  return super.format() + "\nID Number:\t" + idNumber;  }  }  public static class CallingCard extends Card  {  private String cardNumber;  private String pinNumber;    public CallingCard(String s, String car, String pin)  {  super(s);  cardNumber = car;  pinNumber = pin;  }    public String format()  {  return super.format() + "\nCard Number:\t" + cardNumber +  "\nPIN Number:\t" + pinNumber;  }  }  public static class DriverLicense extends Card  {  private int expirationYear;    public DriverLicense(String s, int exp)  {  super(s);  expirationYear = exp;  }    public String format()  {  return super.format() + "\nExpiration Year:\t" + expirationYear;  }  }  **OUTPUT:**  **ID CARD TEST:**  **Card holder: Tyler Quayle**  **ID Number: 950416426**  **CALLING CARD TEST:**  **Card holder: Tyler Quayle**  **Card Number: 1234567890**  **PIN Number: 1988**  **DRIVER LICENSE TEST:**  **Card holder: Tyler Quayle**  **Expiration Year: 2016** |

3.4) Devise another class, Billfold, which contains slots for two cards, card1 and card2, a method void addCard(Card) and a method String formatCards().

The addCard method checks whether card1 is null. If so, it sets card1 to the new card. If not, it checks card2. If both cards are set already, the method has no effect.

Of course, formatCards invokes the format method on each non-null card and produces a string with the format

[card1|card2]

What is your Billfold class?

|  |
| --- |
| public static class Billfold extends Card  {  Card cardOne;  Card cardTwo;    public void addCard(Card input)  {  if(cardOne == null)  {  cardOne = input;  return;  }  if(cardTwo == null)  cardTwo = input;  }    public String formatCards()  {  if(cardOne != null && cardTwo != null)  return "\nCARD ONE:\n" + cardOne.format() +  "\n\nCARD TWO:\n" + cardTwo.format();  else if(cardOne != null)  return "\nCARD ONE:\n" + cardOne.format();  else if(cardTwo != null)  return "\nCARD TWO:\n" + cardTwo.format();  return "ERROR: NO CARDS FOUND!";  }  } |

3.5) Write a tester program that adds two objects of different subclasses of Card to a Billfold object. Test the results of the formatCards methods.

What is the code for your BillfoldTester class?

|  |
| --- |
| Public class BillfoldTester  {  public static void main(String [] args)  {  IDCard id\_test = new IDCard("Tyler Quayle", "950416426");  CallingCard cc\_test = new CallingCard("Tyler Quayle","1230",1988);  Billfold bf\_test = new Billfold();    bf\_test.addCard(id\_test);  bf\_test.addCard(cc\_test);  System.out.println(bf\_test.formatCards());    }  }  **OUTPUT:**  **CARD ONE:**  **Card holder: Tyler Quayle**  **ID Number: 950416426**  **CARD TWO:**  **Card holder: Tyler Quayle**  **Card Number: 1230**  **PIN Number: 1988** |

**I know we were supposed to do in a format of [ card1 | card2 ], however the output looks completely jumbled and I, for one, cannot stand it when it looks like that. So I decided to output my BillFold in a much cleaner design.**

3.6) Explain why the output of your BillfoldTester program demonstrates polymorphism.

**The output of BillfoldTester shows polymorphism because the class of BillFold can take on many different ‘roles’ as it can accept any type of card, ID, DL or Calling card and outputs them using their own inherit format method which in turn calls the superclass of Card;**

3.7) The Card superclass defines a method isExpired, which always returns false. This method is not appropriate for the driver license. Supply a method header and body for DriverLicense.isExpired() that checks whether the driver license is already expired (i.e., the expiration year is less than the current year).

To work with dates, you can use the methods and constants supplied in abstract class Calendar which are inherited by the concrete class GregorianCalendar. You create a Calendar as follows:

GregorianCalendar calendar = new GregorianCalendar();

Then, you can obtain the current year using the constant Calendar.YEAR and method get in GregorianCalendar. The constant indicates that the method should return the current year. By comparing the returned value with the expYear field in DriverLicense, you can determine if the card is expired. The code below will retrieve the current year:

calendar.get(Calendar.YEAR)

|  |
| --- |
| public static class DriverLicense extends Card  {  private int expirationYear;  GregorianCalendar calendar = new GregorianCalendar();    public DriverLicense(String s, int exp)  {  super(s);  expirationYear = exp;  }      public boolean isExpired()  {  if(expirationYear < calendar.get(calendar.YEAR))  return false;    else return true;  }  public String format()  {  return super.format() + "\nExpiration Year:\t" + expirationYear;  }  } |

3.8) The ID card and the phone card don’t expire. What should you do to reflect this fact in your implementation?

**Implement Sa isExpired class into each ID and Calling card that simply returns the super class isExpired. So BillFold can always call cardOne.isExpired() or cardTwo.isExpired();**

|  |
| --- |
| public static class IDCard extends Card  {  private String idNumber;    public IDCard(String s, String id)  {  super(s);  idNumber = id;  }  public boolean isExpired(){return super.isExpired();}  public String format()  {  return super.format() + "\nID Number:\t" + idNumber;  }  }  public static class CallingCard extends Card  {  private String cardNumber;  private String pinNumber;    public CallingCard(String s, String car, String pin)  {  super(s);  cardNumber = car;  pinNumber = pin;  }    public boolean isExpired(){return super.isExpired();}    public String format()  {  return super.format() + "\nCard Number:\t" + cardNumber +  "\nPIN Number:\t" + pinNumber;  }  } |

3.9) Add a method getExpiredCardCount, which counts the number of expired cards in the billfold, to the Billfold class.

|  |
| --- |
| public static class Billfold extends Card  {  Card cardOne;  Card cardTwo;    public void addCard(Card input)  {  if(cardOne == null)  {  cardOne = input;  return;  }  if(cardTwo == null)  cardTwo = input;  }    public int getExpiredCardCount()  {  return (cardOne.isExpired() ? 1 : 0) + (cardTwo.isExpired() ? 1 : 0);  }  public String formatCards()  {  if(cardOne != null && cardTwo != null)  return "\nCARD ONE:\n" + cardOne.format() +  "\n\nCARD TWO:\n" + cardTwo.format();  else if(cardOne != null)  return "\nCARD ONE:\n" + cardOne.format();  else if(cardTwo != null)  return "\nCARD TWO:\n" + cardTwo.format();  return "ERROR: NO CARDS FOUND!";  }  } |

3.10) Write a BillfoldTester class that populates a billfold with a phone calling card and an expired driver license. Then call the getExpiredCardCount method. Run your tester to verify that your method works correctly.

What is your tester class?

|  |
| --- |
| Public class BillfoldTester  {  public static void main(String [] args)  {  CallingCard cc\_test =  new CallingCard("Tyler Quayle", "1230", "1988");  DriverLicense dl\_test = new DriverLicense("Tyler Quayle", 2016);  Billfold bf\_test = new Billfold();    bf\_test.addCard(cc\_test);  bf\_test.addCard(dl\_test);  System.out.println(bf\_test.formatCards());  System.out.println("\nExpired Cards: ” +  "bf\_test.getExpiredCardCount());  }  }  **OUTPUT:**  **CARD ONE:**  **Card holder: Tyler Quayle**  **Card Number: 1230**  **PIN Number: 1988**  **CARD TWO:**  **Card holder: Tyler Quayle**  **Expiration Year: 2016**  **Expired Cards: 1** |