CS 535 Object-Oriented Programming & Design
Fall Semester, 2000
Doc 13 Assignment 1 Comments
Contents

References

Student papers

Assignment 1 Comments

Total points 40
24 points for working program
16 points for design, style, code

Score	# of students
40	1
39	1
38	2
37	8
36	8
35	7
34-30	14
29-25	5
24-0	2

Need for Comments

```
abstract class Money {
   protected double amount;
   protected String nation;
```

amount is
- in US currency?
- native currency?
- in smallest currency type (cents, farthings, etc)?
- in largest currency type (dollar, pounds, marks, etc)?

nation format is
- two letter country code?
- Full name of country?
- Name of currency class?

```
public abstract class Money {
   private int[] amount = new int[5] //amount money
```

Why 5?
What is each element of the array for?

Need for Comments

```
public abstract class Currency {

   public static Currency getInstance(String newCurrency) {
      blah
   }
```

Format of newCurrency?

```
public class BEG extends Currency {

   // un-instantiable constructor
   private BEG() { }
```

Why?

```
public class Currency {
   double exchangeRate;
   double standard;
```

Standard what?

Need for Comments

```
public interface Currency {

   /**
    * Change the currency to lower format
    * @return float, a float number after change
   public int lowerFormat();

   /**
    * Change the currency to upper format
    * @return float, a float number after change
   public float upperFormat();
```

But what is upper and lower format?

Exceptions

```
public BEMoney divide(int divisor) {
   if ( divisor == 0 ) {
      System.out.println( "Error can't divide by zero");
      return this;
   }
   do the real work
```

Don't use print statements in Library code unless for debugging

If reach an error, do not continue throw exception

```
public BEMoney divide(int divisor) {
   if ( divisor == 0 ) {
      throw new ArithmeticException("Zero divide");
   }
   do the real work
```

Use is-kind-of Inheritance

```
public class US {

   stuff here
}

public class BE extends US {
   more stuff
}
```

BE is not a kind of US currency

Fields as Parameters

```
public class USCurrency extends Currency {
   private long dollars;
   private long cents;
   private double value;
   private double conversionRate;

   public USCurrency( double universalValue, double conversionRate) {
      cents = dollars = 0L;
      value = universalValue;
      this.conversionRate = conversionRate;
      computeValues();
   }
   //page of code removed

   private void computeValues() {
      cents =(long) value * conversionRate* 100;
      reduce();
   }
   //page of code removed

   private void reduce() {
      dollar += cents / 100;
      cents %= 100;
   }
```

value and conversionRates are fields used as parameters

computeValues() has side effects that are not clear

A Clearer Version

```
public class USCurrency extends Currency {
   private long dollars;
   private long cents;

   public USCurrency( double universalValue, double conversionRate) {
      long totalCents = universalToCents(universalValue,
conversionRate);
      dollars = dollarsIn( totalCents);
      cents = centsMinusDollars( totalCents);
   }

   private long universalToCents(double universalValue,
         double conversionRate) {
      return (long) universalValue * conversionRate * 100;
   }

   private long dollarsIn(long totalInCents) {
      return (long) totalInCents/ 100;
   }

   // needs better name
   private long centsMinusDollars(long totalInCents) {
      return totalInCents - dollarsIn( totalInCents) * 100 ;
   }
```

In this example the action of the constructor is much clearer

No side effects

Double State

```
public abstract class Currency {
   private int value;

   public Currency( int totalValue ) {
      value = totalValue;
   }

public class USCurrency extends Currency {
   private int dollars;
   private int cents;

   public USCurrency(int dollars, int cents ) {
      super( cents + dollars * 100);
      this.dollars = dollars;
      this.cents = cents;
   }
```

The state is duplicated

Make for more work to maintain same values in two different locations

Causes errors - when duplicated states are not correctly updated

One interface with one implementation

```
public interface Money {
   public Money add( Money toAdd);
   public Money subtract( Money toSubtract);
   // other method in assignment removed
}

public abstract class Currency implement Money {
      public Money add( Money toAdd);
      public Money substract( Money toSubtract ) {
         return add( toSubtract.multiple( -1));
      }
}

public class USCurrency extends Currency {
   code removed
}
```

Why have the Money interface?

Positional Data

```
public abstract class Currency {

   double[] exchangeRate = {0.193, 0.238, 4.866, 1.0 };

   etc.
```

Which element of exchangeRate belongs to which currency?

Error prone

Information Hiding

```
public class Currency {
   int value[];

   // stuff removed

   public int[] add(int[] amount ) {
      // add the amount to value
```

What is Currency?

```
public class UKCurrency {
   int valueInUSCents

   public UKCurrency(int amount ) {
      valueInUSCents = amount;
   }
etc.
}
```

Keep internal representation internal

```
public class UKCurrency {
   int valueInUSCents

   public UKCurrency(int amountInPounds ) {
      valueInUSCents = amount * POUNDS_TO_DOLLARS;
   }
```

Cut and Paste

```
public String toString() {
   return UKP + " UKP " + UKS + " UKS " +
      UKE + " UKE " + UKF + " UKF ";
   }

public void display() { // for debugging
   System.out.println( UKP + " UKP " + UKS + " UKS " +
      UKE + " UKE " + UKF + " UKF ");
```

Why not call toString()?

More Cut and Paste

```
public Money multiply(double n ) {
   int totalInBaseUnits =
      UKP * 20* 12* 4+ UKS *12 * 4 + UKE * 4 + UKF;
   UKMoney result =
      new UKMoney( 0, 0, 0, (int)(totalInBaseUnits * n) );
   return result;
}

public Money divide(double n ) {
   int totalInBaseUnits =
      UKP * 20* 12* 4+ UKS *12 * 4 + UKE * 4 + UKF;
   UKMoney result =
      new UKMoney( 0, 0, 0, (int)(totalInBaseUnits / n) );
   return result;
}

public Money add( Money toAdd) {
   int totalInBaseUnits =
      UKP * 20* 12* 4+ UKS *12 * 4 + UKE * 4 + UKF;
   UKMoney ukToAdd = toAdd.convertToUK():
      etc.
}
```

Why not add a function?

```
private int totalInBaseUnits() {
   return UKP * 20* 12* 4+ UKS *12 * 4 + UKE * 4 + UKF;
}
```

More Cut and Paste

```
public class UKCurrency {
   public Currency add( Currency toAdd) {
      USCurrency asUS = toAdd.toUS();
      UKCurrency ukToAdd = fromUS( asUS);
      int total = this.value + ukToAdd.value;
      return new UKCurrency( total );
   }

   public Currency subtract( Currency toSubtract) {
      USCurrency asUS = toSubtract.toUS();
      UKCurrency ukToSubtract = fromUS( asUS);
      int total = this.value - ukToSubtract.value;
      return new UKCurrency( total );
   }
}

public class DECurrency {
   public Currency add( Currency toAdd) {
      USCurrency asUS = toAdd.toUS();
      DECurrency deToAdd = fromUS( asUS);
      int total = this.value + deToAdd.value;
      return new DECurrency( total );
   }

   public Currency subtract( Currency toSubtract) {
      USCurrency asUS = toSubtract.toUS();
      DECurrency deToSubtract = fromUS( asUS);
      int total = this.value - deToSubtract.value;
      return new DECurrency( total );
   }
}
```

A lot of repeated structure

Many Methods

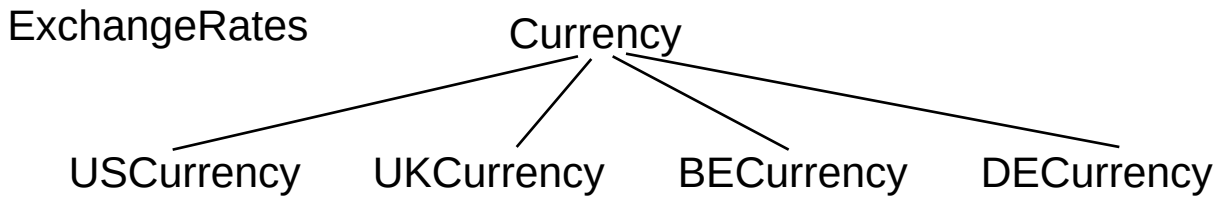
```
public class UKCurrency {
   int valueInUSCents

   public Currency toUS() {
      //blah
   }

   public Currency toUK() {
      return this;
   }

   public Currency toDE() {
      //blah
   }

   public Currency toBE() {
      //blah
   }
```

Repeated in each Currency class

Adding a new currency type means adding methods to all the existing currency classes, which we would like to avoid

Some Solution

Classes



How to store amount in Currency objects

- In native currency

  Direct and clear

- In smallest native units (cents, farthings, etc)

  Simplifies computations

- In universal currency (US?)

  Simplifies many computations
  Requires careful use of currency exchange rate
  Most got the exchange rate incorrectly

Will use smallest native units

multiply, divide, add, subtract

Will make them return a new currency object and not change receiver

How to avoid duplicating code in these methods?

Reuse methods

```
package currency;

public abstract class Currency
  {
  private int amount; // In smallest units of native currency

  public abstract Currency add(Currency toAdd);
  public abstract Currency multiply(float multiplier);
  public Currency subtract( Currency toSubtract)
    {
    return add( toSubtract.multiply( -1));
    }

  public  Currency divide(float divisor) throws ArithmeticException
    {
    if ( divisor == 0 )
      {
      throw new ArithmeticException("Zero divide");
      }
    return multiply( 1/divisor);
    }
  }
```

multiply

```
public  class USCurrency extends Currency
   {

   public  Currency multiply(float multiplier)
      {
      USCurrency result = new USCurrency();
      result.amount = (int) Math.floor(amount * multiplier);
      return result;
      }
   }
```

This leads to nearly identical methods in each Currency subclass

Only repeat different code

Parent class contains the algorithm
Child classes only add the methods that are different

```
public abstract class Currency
  {
  protected int amount; // In smallest units of native currency

  public abstract Currency getNewInstance();
  public  Currency multiply(float multiplier)
    {
    Currency result = getNewInstance();
    result.amount = (int) Math.round(amount * multiplier);
    return result;
    }
  }

public  class USCurrency extends Currency
  {
  public  Currency getNewInstance()
    {
    return new USCurrency();
    }
  }

public  class BECurrency extends Currency
  {
  public  Currency getNewInstance()
    {
    return new BECurrency();
    }
```

Java Trick - Use Clone

```
public abstract class Currency implements Cloneable
  {
  protected int amount; // In smallest units of native currency

  public  Currency multiply(float multiplier)
    {
    try
      {
      Currency result =(Currency) this.clone();
      result.amount = (int) Math.round(amount * multiplier);
      return result;
      }
    catch (CloneNotSupportedException cantHappen)
      {
      throw new ArithmeticException("Clone exception inmultiply");
      }
    }
  }
```

No need for any method in subclasses to support multiply

Implementing add

```
public abstract class Currency
  {
  protected int amount; // In smallest units of native currency

  /**
   * @param amount in counties largest currency unit
   */
  protected abstract Currency getNewInstance(double amount);

  public abstract Currency convertTo( Currency   aCurrency);

  public  Currency add(Currency toAdd)
    {
    Currency converted = toAdd.convertTo( this);
    Currency result = getNewInstance(0.0);
    result.amount = amount + converted.amount;
    return result;
    }
```

Implementing convertTo

```
public abstract class Currency
  {
  /**
   * In smallest units of native currency
   */
  protected int amount;

  public abstract String getCurrencyCode();

  /**
   * @param amount in counties largest currency unit
   */
  protected abstract Currency getNewInstance(double amount);

  /**
   * Return the amount in terms of the countries largest currency unit
   */
  protected abstract double getAmount();
```

Implementing convertTo

```
public  Currency convertTo( Currency aCurrency) throws
   CurrencyConversionException
   {
   String fromCountry = getCurrencyCode();
   String toCountry = aCurrency.getCurrencyCode();
   if (fromCountry.equals( toCountry))
      return this;
   try
      {
      ExchangeRates currentRates = ExchangeRates.getInstance();
      double convertRate =
         currentRates.convertFromTo( fromCountry, toCountry);
      double convertedAmount = getAmount() * convertRate;
      return aCurrency.getNewInstance(convertedAmount);
      }
   catch (java.io.IOException error)
      {
      throw new  CurrencyConversionException(
         "could not find exchange rate data");
      }
   }
```

ExchangeRates

```
package currency;
import sdsu.io.SimpleFile;
import java.io.IOException;
import sdsu.util.LabeledData;
import java.util.Enumeration;

public class ExchangeRates
   {
   private static final String RATE_TABLE = "rateTable";
   private static ExchangeRates instance;

   LabeledData rates = new LabeledData();

   public static ExchangeRates getInstance() throws IOException
      {
      if (instance == null )
         instance = new ExchangeRates( RATE_TABLE );
      return instance;
      }

   // Force user to use static getInstance() so only one object is created
   private ExchangeRates(String fileName) throws IOException
      {
      readRatesFromFile( fileName);
      }
```

ExchangeRates Continued

```
public double convertFromTo( String fromCurrency,
   String toCurrency )
   {
   return toUSFrom(fromCurrency) * fromUSTo( toCurrency) ;
   }

private void readRatesFromFile( String fileName) throws IOException
   {
   SimpleFile rateFile = new SimpleFile( fileName);
   String asciiRateData = rateFile.getContents();
   rates.fromString( asciiRateData);
   Enumeration keys = rates.keys();
   while (keys.hasMoreElements() )
      {
      String key = (String) keys.nextElement();
      String value =(String) rates.get( key);
      rates.put( key, Double.valueOf( value));
      }
   }

private double fromUSTo( String currency)
   {
   return 1/toUSFrom( currency);
   }

private double toUSFrom( String currency)
   {
   return ((Double)rates.get( currency )).doubleValue();
   }
}
```

USCurrency

```
package currency;

public  class USCurrency extends Currency {
   private static final String currencyCode = "USD";

   public USCurrency() { this( 0, 0); }

   public USCurrency(int dollars, int cents) {
      amount = dollars * 100 + cents;
   }

   public String getCurrencyCode() { return currencyCode; }

   public int getDollars() { return amount / 100; }

   public int getCents() { return (amount % 100); }

   // returns amount in dollars
   protected double getAmount() {
      return getDollars() + getCents()/100.0;
   }

   protected Currency getNewInstance(double amountInDollars) {
      return new USCurrency(0,(int) Math.floor(amountInDollars * 100));
   }
}
```

Formatted oddly to fit on one page

DECurrency

```
package currency;

public  class DECurrency extends Currency {
   private static final String currencyCode = "DEM";

   public DECurrency() { this( 0, 0); }

   public DECurrency(int marks, int pfennigs) {
      amount = marks * 400 + pfennigs;
   }

   public String getCurrencyCode() { return currencyCode; }

   public int getMarks( ){ return amount / 400; }

   public int getPfennigs() { return (amount % 400); }

   // returns amount in marks
   protected double getAmount() {
      return getMarks() + getPfennigs()/400.0;
      }

   protected Currency getNewInstance(double amountInMarks) {
      return new DECurrency(0,(int) Math.floor(amountInMarks * 400));
   }
}
```

Formatted oddly to fit on one page

DECurrency verses USCurrency

The two classes only deal with local currency issues

The two classes are very similar

- DECurrency was created by cut-paste from USCurrency

Should be able to combine into one class!

- Left as an exercise for the reader

What about BECurrency & UKCurrency?

Further Issues

How to implement convertTo(String currencyType) without if/case statement

Reasonable process to convert strings to currency objects