Formatting & Design

GEEN163

"Debugging is twice as hard as writing the code in the first place. Therefore, if you write the code as cleverly as possible, you are, by definition, not smart enough to debug it."

Brian W. Kernighan

Teaching Evaluation

- The official University teaching evaluation survey is on Blackboard
- Complete the survey for ALL classes

No Recitation Next Week

- There will be no recitation classes the week of Thanksgiving

format Method

- The java.io.PrintWriter class has two identical methods, format and printf, to format output

  format(String form, var1, var2, …)

- Writes the variables to the output as specified by the form string
- Very similar to printf in the C programming language

java.io.PrintWriter

- The java.io.PrintWriter class is commonly used to write data to a file or the screen
- System.out is an object of the class java.io.PrintWriter
- You can call the format and printf methods on System.out

```
System.out.printf("A piece of pie %5.3\n", Math.PI);
```

Format Descriptors

- The form string may contain text with descriptors located in it.
- The descriptors start with a percent sign, % followed optionally by a length and then a format type character

format	data type	result
'd'	int or long	The result is formatted as a decimal integer
'f'	double or float	The result is formatted as a decimal number
‘s’	String	The string

Output length

- You can specify a number between the % and the descriptor character to indicate the minimum number of characters to display
- You can specify the maximum number of digits to the right of the decimal point

%*minlength*.*maxprecision***f**

Format Examples

```
double   e = 2.718281828459045;
System.out.format("answer is %5.3f", e);
```

will display “2.718”

```
System.out.format("answer is %8.4f", e);
```

will display “ 2.7183”

Creating Columns

- You can specify a number between the % and the descriptor character to indicate the minimum number of characters to display
- This is useful to display strings or numbers in a column

%*minlength***s** *or* **%***minlength***d**

Formatting Integers

```
for (int num = 1; num < 10000; num*=17) {
    System.out.format("num %4d and %6d num squared \n",
                                    num, num*num);
}
```

generates the output

```
num    1 and      1 num squared
num   17 and    289 num squared
num  289 and  83521 num squared
num 4913 and 24137569 num squared
```

Newline Character

- Like the System.out.print method, format and printf do no automatically end with a new line
- You can put the new line character `'\n'` in the format string to specify where you want the output to go to a new line

What is displayed?

```
double x = 10.0;
double y = x * 2.0 / 3.0;
System.out.printf("y is %6.2f", y);
```

A. y is 10.00

B. y is 6.66

C. y is 6.67

D. y is 6.66667

E. y is 6.666666666667

Bank Account Example

- Consider a class that holds information about a bank account
- A bank account object will need to contain
  - name of the account owner
  - balance of the account

Example Bank Account Class

```
public class BankAccount {
    private double money;       // balance in the account
    private String owner;       // name of the owner
        // constructor with two parameters
    public BankAccount(double initial, String name){
        money = initial;        // save initial balance
        owner = name;           // save name of owner
    }
       // default constructor
    public BankAccount() {
        money = 0.00;
        owner = "unknown";
    }
}
```

More Example Methods

```
/*  method to add money to the balance */
   public void deposit(double cash) {
     money = money + cash;
  }

/*  method to remove money from the balance */
   public void withdraw(double cash) {
     money = money - cash;
  }
```

Method with an Object Parameter

```
public void transfer(
                BankAccount you,
                double      amount) {

    you.money -= amount;
    money     += amount;
}
```

Write with your team

- Create two objects of the BankAccount class with your name and some initial balance
- Add $500 to one of the accounts
- Subtract $1.00 from the other account
- Move $25 from one account to the other

Using the BankAccount class

```
BankAccount mine  = new
    BankAccount( 10000.00, "Joe");
BankAccount yours = new
      BankAccount( 47.50, "Fred");

mine.deposit(500.00);
yours.withdraw( 1.00 );
mine.transfer( yours, 25.00 );
```

How can you avoid negative balances?

```
public void transfer(
                BankAccount you,
                double      amount) {

    you.money -= amount;
    money     += amount;
}
```

Modify the method to not change the money if this will make it negative

Possible Solution

```
public void transfer(
                BankAccount you,
                double      amount) {

    if (you.money >= amount) {
        you.money -= amount;
        money     += amount;
    }
}
```

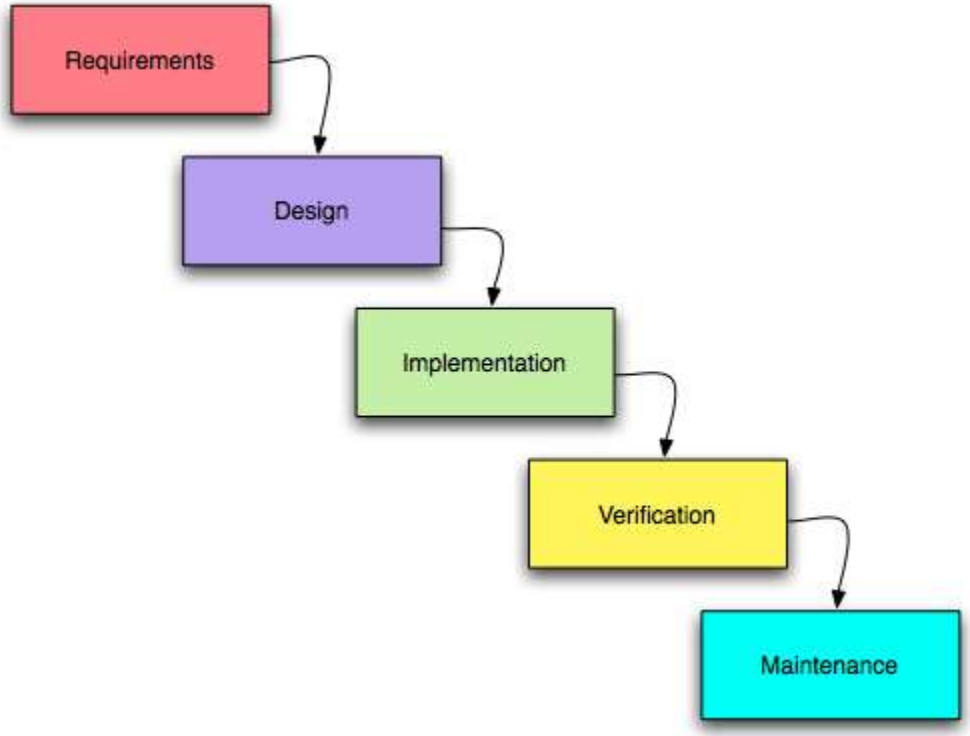
Software Engineering Steps

- **Requirements** – Define exactly what the program is supposed to do from the user's point of view
- **Design** – How will the program run
- **Implementation** – Write the Java
- **Verification** – Test the program looking for any errors
- **Maintenance** – Fix the errors you missed the first time

Waterfall Model

- The waterfall model is one software engineering technique for creating programs

- There are several other software engineering processes



Algorithm

- An algorithm is a step-by-step procedure for solving a problem
- Algorithms can be expressed in several ways
  - English
  - Flow charts
  - Formal logic
  - pseudo code
- An algorithm is less detailed than a program, but provides all the information necessary to solve the problem

What should be done first?

A. Write Java code
B. Write a design document
C. Talk with the customer
D. Fix any bugs

Program Description

- Given the cost of a purchase and the amount paid, determine how many and what coins to give in change

Write an algorithm

- Briefly explain how a program could determine the coins to return
- Assume the program has read the cost and amount paid and calculated the change = paid - cost

Algorithm for Change

- Read the cost and amount paid
- Calculate the amount of change
- For each coin from largest to smallest
- While the change is more or equal to the coin
  - Give a coin
  - subtract the coin from the change

Translating Algorithm to Java

- How should the data be represented?
- Since money is represented as a number with decimal digits, we can use doubles
- Variables needed are:
  - cost = input from user
  - paid = input from user
  - change = paid – cost
  - array of coin values

Initial Data Declarations

```
double cost;        // input cost
double paid;        // input amount paid
double change;      // amount to be returned
double[] coin = {0.25, 0.10, 0.05, 0.01};    // coins
```

- We may find we need more variables later

Input Algorithm

- Read the cost and amount paid
- Calculate the amount of change

```
cost = keyboard.nextDouble();
paid = keyboard.nextDouble();
change = paid – cost;
```

For all coins

- For each coin from largest to smallest

- Since the array is of fixed size, a **`for`** loop would be appropriate

```
for (int denom = 0; denom < coin.length; denom++)
```

Calculate Change for a Coin

- While the change is more or equal to the coin
  - Give a coin *(increment a counter)*
  - subtract the coin from the change

```
int count = 0;
while ( change >= coin[denom] ) {
        change -= coin[denom];
        count++;
}
System.out.println(count+" "+coin[denum]);
```

Full Program

```
public class ChangeDouble {
 public static void main(String[] unused) {
   double[] coin = {0.25, 0.10, 0.05, 0.01};   // coins
   double cost, paid;                          // input cost
   double change;                              // amount to return
   java.util.Scanner keyboard = new
                          java.util.Scanner(System.in);
   System.out.print("Enter the cost and amount paid >");
   cost = keyboard.nextDouble();               // read input
   paid = keyboard.nextDouble();
   change = paid - cost;
   for (int denom = 0; denom < coin.length; denom++) {
      int count = 0;
      while (change >= coin[denom]) {
         change = change - coin[denom];
         count++;
      }
      System.out.println(count+" "+coin[denom]+" cent ");
   }
}}
```

Testing

- What are good test values?

- What will the program do if incorrect values are given?

What will the program do if the cost is greater than the amount paid?

A. Show 0 for all coins
B. Infinite loop
C. Show negative number of coins
D. none of the above

Enhancements

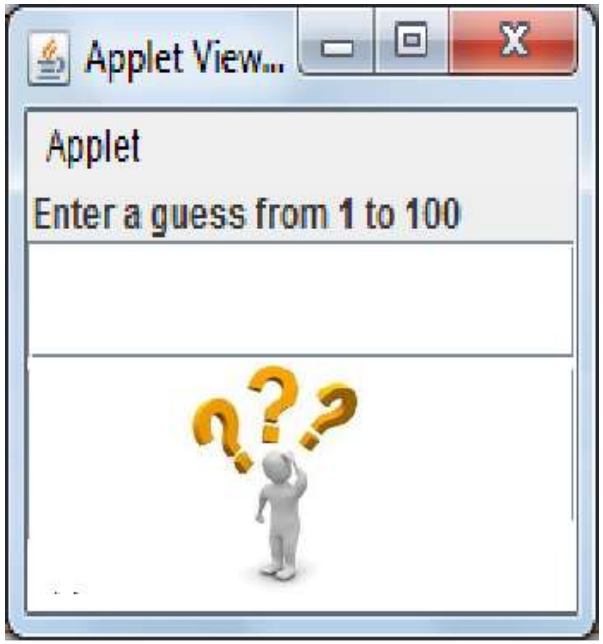
- How might we improve the output by identifying the coins by name (e.g. “dime”)?

- What would it take to provide change for Euros

1c, 2c, 5c, 10c, 20c, 50c, €1, €2

Pictures in GUIs

- Sometimes it is nice to display the contents of a graphics file in your program
- A JLabel object can contain text or an image from a file, such as .gif or .jpg



javax.swing.ImageIcon

- An ImageIcon object can hold the image from a graphics file
- You can specify the graphics file as a parameter to the constructor

```
javax.swing.ImageIcon   cow  = new
        javax.swing.ImageIcon("dir/picture.gif");
```

Putting Images in JLabels

- A JLabel constructor can take an ImageIcon
- The setIcon( ) method of JLabel or JButton will display an ImageIcon object

```
ImageIcon   cow  = new ImageIcon("dir/picture.gif" );
ImageIcon   bull  = new ImageIcon("dir/photo.jpg" );

JLabel  goat = new JLabel( bull );
goat.setIcon( cow );
```

Schedule

Monday, November 18 Secure Programming Lab	Wednesday, November 20 Programming practice Quiz Lab	Friday, November 22 review Quiz
Monday, November 25 **Exam 3**	Wednesday, November 27 *Thanksgiving Holiday* *(no classes)*	Friday, November 29 *Thanksgiving Holiday* *(no classes)*
Monday, December 2 Software engineering Lab	Wednesday, December 4 review Final Lab	Final

No Recitation Next Week

- There will be no recitation classes the week of Thanksgiving