

TEAM C

JAVA ASSIGNMENT -2

Team Members:

Harshit Oberoi

Tanya Khullar

Yi Qi

zhenzhouyu

Yu Gong

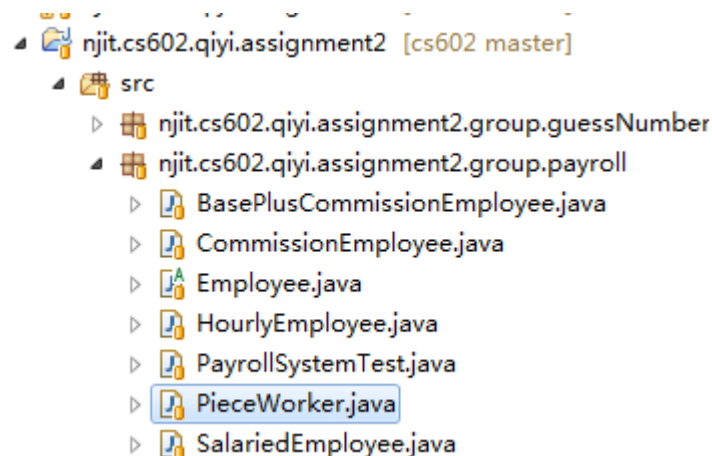
XiuruiHou

Xuan zhang

1. Payroll system

This problem is to ask us to add a new kind of employee to the original payroll system. We need to create a new subclass for that employee and implement its own unique earning behavior, string representation and corresponding auxiliary attributes and methods. Also we need modify the original test class to test the new employee. This problem can be done by the following steps.

- a. We need to create a new subclass for piece worker which extends the super class Employee



- b. We need to create some private attributes for this class and also do the validation in the constructor

```
public class PieceWorker extends Employee {
    private double wage; // wage per piece
    private double pieces; // number of pieces produced

    public PieceWorker(String firstName, String lastName,
        String socialSecurityNumber, double wage, double pieces) {
        super(firstName, lastName, socialSecurityNumber);
        // TODO Auto-generated constructor stub

        if (wage < 0.0) // validate wage
            throw new IllegalArgumentException(
                "wage per piece must be >= 0.0");
        this.wage = wage;
        this.pieces = pieces;
    }
}
```

- c. Implement corresponding get and set method. Note that the set method also need do validation.

```
public double getWage() {
    return wage;
}

public void setWage(double wage) {
    if (wage < 0.0) // validate wage
        throw new IllegalArgumentException(
            "wage per piece must be >= 0.0");
    this.wage = wage;
}

public double getPieces() {
    return pieces;
}

public void setPieces(double pieces) {
    this.pieces = pieces;
}
```

- d. Implement the earnings and toString method, in the toString method, we first call its parent's toString method and then add its own information behind that.

```
@Override
public double earnings() {
    // TODO Auto-generated method stub
    return wage * pieces;
}

@Override
public String toString(){
    return String.format("piece worker: %s\n%s: $%,.2f; %s: $%,.2f",
        super.toString(), "wage per piece", getWage(),
        "pieces produced", getPieces());
}
```

e. In the test class, we also need to add our new employee.

```
"Bob", "Lewis", "444-44-4444", 5000, .04, 300);
PieceWorker pieceWorker = new PieceWorker("Yi", "Qi", "555-55-5555", 1.5, 1000);

System.out.println("Employees processed individually:");

System.out.printf("%n%s%n%s: $%,.2f%n%n",
    salariedEmployee, "earned", salariedEmployee.earnings());
System.out.printf("%n%s%n%s: $%,.2f%n%n",
    hourlyEmployee, "earned", hourlyEmployee.earnings());
System.out.printf("%n%s%n%s: $%,.2f%n%n",
    commissionEmployee, "earned", commissionEmployee.earnings());
System.out.printf("%n%s%n%s: $%,.2f%n%n",
    basePlusCommissionEmployee,
    "earned", basePlusCommissionEmployee.earnings());
System.out.printf("%n%s%n%s: $%,.2f%n%n",
    pieceWorker,
    "earned", pieceWorker.earnings());

// create four-element Employee array
Employee[] employees = new Employee[5];

// initialize array with Employees
employees[0] = salariedEmployee;
employees[1] = hourlyEmployee;
employees[2] = commissionEmployee;
employees[3] = basePlusCommissionEmployee;
employees[4] = pieceWorker;
```

f. Make a small modification for the last print, so that we won't print the package name

```
// get type name of each object in employees array
for (int j = 0; j < employees.length; j++)
    System.out.printf("Employee %d is a %s%n", j,
        employees[j].getClass().getSimpleName());
} // end main
```

g. The final running result:

```
<terminated> PayrollSystemTest [Java Application] C:\Program Files\Java\jdk1.8.0_60\bin\javaw.exe (2016年3月21)
Employees processed individually:

salaried employee: John Smith
social security number: 111-11-1111
weekly salary: $800.00
earned: $800.00

hourly employee: Karen Price
social security number: 222-22-2222
hourly wage: $16.75; hours worked: 40.00
earned: $670.00

commission employee: Sue Jones
social security number: 333-33-3333
gross sales: $10,000.00; commission rate: 0.06
earned: $600.00

base-salaried commission employee: Bob Lewis
social security number: 444-44-4444
gross sales: $5,000.00; commission rate: 0.04; base salary: $300.00
earned: $500.00

piece worker: Yi Qi
social security number: 555-55-5555
wage per piece: $1.50; pieces produced: 1,000.00
earned: $1,500.00

Employees processed polymorphically:

salaried employee: John Smith
social security number: 111-11-1111
weekly salary: $800.00
earned $800.00

hourly employee: Karen Price
social security number: 222-22-2222
hourly wage: $16.75; hours worked: 40.00
earned $670.00

commission employee: Sue Jones
social security number: 333-33-3333
gross sales: $10,000.00; commission rate: 0.06
earned $600.00

base-salaried commission employee: Bob Lewis
social security number: 444-44-4444
gross sales: $5,000.00; commission rate: 0.04; base salary: $300.00
new base salary with 10% increase is: $330.00
earned $530.00

piece worker: Yi Qi
social security number: 555-55-5555
wage per piece: $1.50; pieces produced: 1,000.00
earned $1,500.00

Employee 0 is a SalariedEmployee
Employee 1 is a HourlyEmployee
Employee 2 is a CommissionEmployee
Employee 3 is a BasePlusCommissionEmployee
Employee 4 is a PieceWorker
```

2. Guess Number

This problem is asking us to implement a simple guess number game. The rule of the game is very simple, for each round, we just need to generate a number to guess and after each guessing we need to make a comparison and give corresponding hint and change the color of background. This problem can be done by the following steps.

- a. Create a class for this game and extends JFrame. To make this game a little general, we do not limit the upper bound of the number to 1000, we allow the caller to pass an upper bound to this class although we have a default upper bound 1000.

```
/**
 * constructor
 * @param max the upper bound of the number to be guessed
 */
public GuessNumber(int max){
    super();
    this.max = max;
    init();
    play();
}

public GuessNumber(){
    this(1000);
}
```

- b. Define all the attributes. We need to store the upper bound, the current secret number and last number the player guessed (so that we can give a better hint). Also, we divide the JFrame into two JPanels, one is for show the hint and restart button, and the other one is for input guessing

number.

```
public class GuessNumber extends JFrame {  
  
    private static final long serialVersionUID = 1L;  
    private int max; //the upper bound of the number to be guessed  
    private int secretNumber; // the number to be guessed  
    private int lastNumber; // last number that was guessed  
  
    private JPanel toolbar;  
    private JButton restart;  
    private JLabel numberRange;  
    private JLabel lastN;  
    private JLabel guessingResult;  
  
    private JPanel guessingPanel;  
    private JLabel textLabel;  
    private JTextField textField;  
    private JButton guess;
```

- c. For the first panel, we add all the components to it, including a “restart” button and corresponding action, a “number range” label to show the number range of the guessing number, a “lastN” label to show the last guessing number, and a “guessing result” label to show last guessing result.

```
private void init(){  
    this.setSize(600, 300);  
    this.setTitle("Guess Number");  
    this.setDefaultCloseOperation(EXIT_ON_CLOSE);  
  
    toolbar = new JPanel(new GridLayout(1, 4, 5, 0));  
    add(toolbar, BorderLayout.NORTH);  
  
    restart = new JButton("Play Again");  
    restart.addActionListener(new ActionListener() {  
  
        @Override  
        public void actionPerformed(ActionEvent e) {  
            // TODO Auto-generated method stub  
            play();  
        }  
    });  
    toolbar.add(restart);  
  
    numberRange = new JLabel("Number Range: " + "1 - " + max);  
    numberRange.setHorizontalAlignment(SwingConstants.CENTER);  
    toolbar.add(numberRange);  
  
    lastN = new JLabel();  
    lastN.setHorizontalAlignment(SwingConstants.CENTER);  
    toolbar.add(lastN);  
  
    guessingResult = new JLabel();  
    guessingResult.setHorizontalAlignment(SwingConstants.CENTER);  
    toolbar.add(guessingResult);
```

- d. For the second panel, we need to add the input text field and corresponding label and button. For the “guess” button, we need to add corresponding actions: first we need to validate the user input and then check the guessing result and update the corresponding label and the color of background. If the player gets the correct answer, we also need to set the input field not editable and unable the button.

```
guessingPanel = new JPanel(new FlowLayout(FlowLayout.LEFT));
guessingPanel.setOpaque(true);
add(guessingPanel);

textLabel = new JLabel("Number:      ");
guessingPanel.add(textLabel);

textField = new JTextField(20);
guessingPanel.add(textField);

guess = new JButton("Guess");
guess.addActionListener(new ActionListener() {

    @Override
    public void actionPerformed(ActionEvent e) {
        // TODO Auto-generated method stub
        int number = 0;
        try{
            number = Integer.parseInt(textField.getText());
            if (number <= 0 || number > max) throw new Exception("Invalid input");
            setBackground(number);
            setLabelContent(number);
            lastNumber = number;
            if (number == secretNumber){
                JOptionPane.showMessageDialog(rootPane, "Correct");
                guessingPanel.setBackground(Color.WHITE);
                textField.setEditable(false);
                guess.setEnabled(false);
            }
        }
        catch(Exception ex){
            JOptionPane.showMessageDialog(rootPane, "Invalid input");
        }
    }
});
guessingPanel.add(guess);
this.setVisible(true);
```

- e. When restarting the game, we need get a new random number and reset all the information including background,

labels, and make the input field editable and enable the button.

```
private void play(){  
    // get a new secret number  
    setSecretNumber();  
    // reset  
    this.guessingPanel.setBackground(Color.WHITE);  
    this.guessingResult.setText("Have a try");  
    this.lastN.setText("Last Number:");  
    textField.setText("");  
    textField.setEditable(true);  
    guess.setEnabled(true);  
}  
// calculate and set the number to be guessed  
private void setSecretNumber(){  
    secretNumber = (int)(Math.random() * max) + 1;  
}
```

- f. Some details for setting the background and labels. If the guessing number is closer to the secret number than the last guessing, change the color of the background of the guessing panel to red; otherwise change it to blue.

For the result label, just make a simple comparison, and update the text of the label.

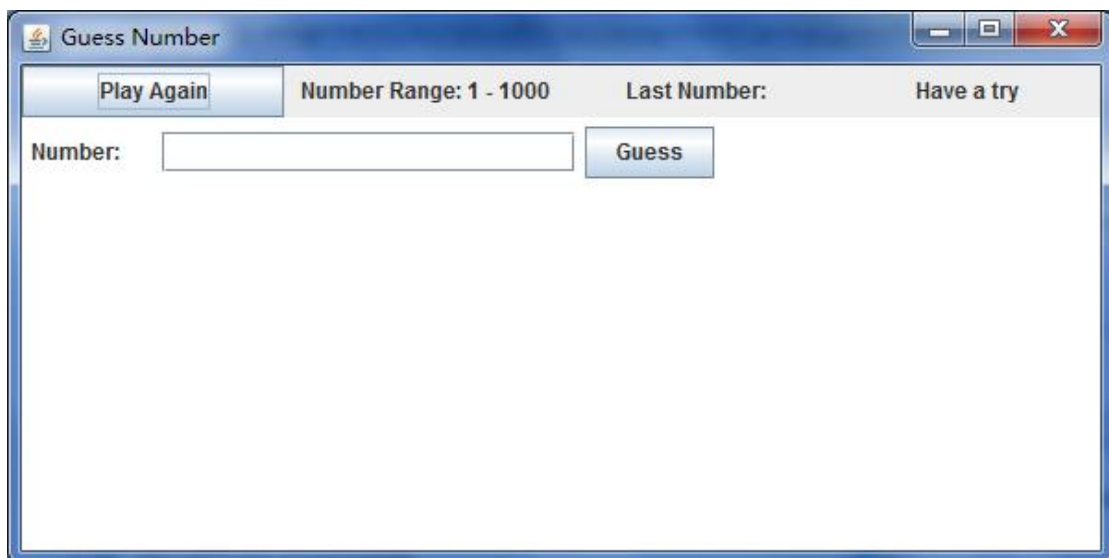
If the input number is valid guessing number, update the last number label. (The only situation that this method can get an invalid input is in the initiation phase, in which we don't have a last number)

```
// set the color of background
private void setBackgroundColor(int number){
    if (Math.abs(number - secretNumber) >= Math.abs(lastNumber - secretNumber)){
        //get colder
        this.guessingPanel.setBackground(Color.BLUE);
    }
    else{
        // get warmer
        this.guessingPanel.setBackground(Color.RED);this.setVisible(true);
    }
}

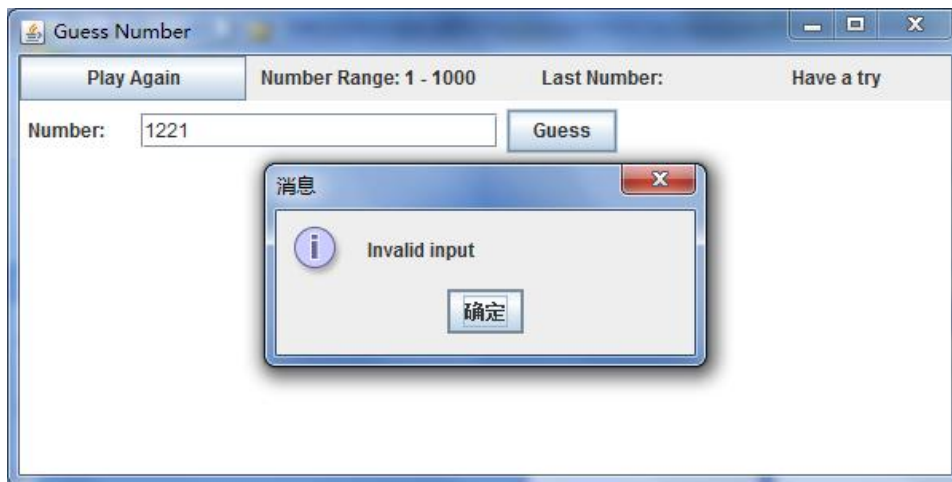
// set the content of last number and result label
private void setLabelContent(int number){
    if (number == secretNumber) this.guessingResult.setText("Correct");
    else if (number > secretNumber) this.guessingResult.setText("Too High");
    else this.guessingResult.setText("Too Low");
    if (number > 0) lastN.setText("Last Number: " + String.valueOf(number));
}
```

g. The final running result

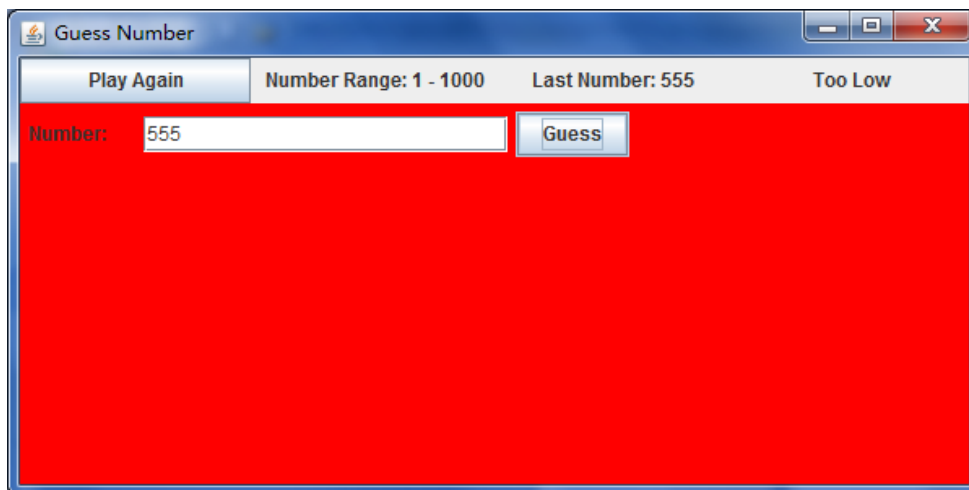
When starting, it will like this:



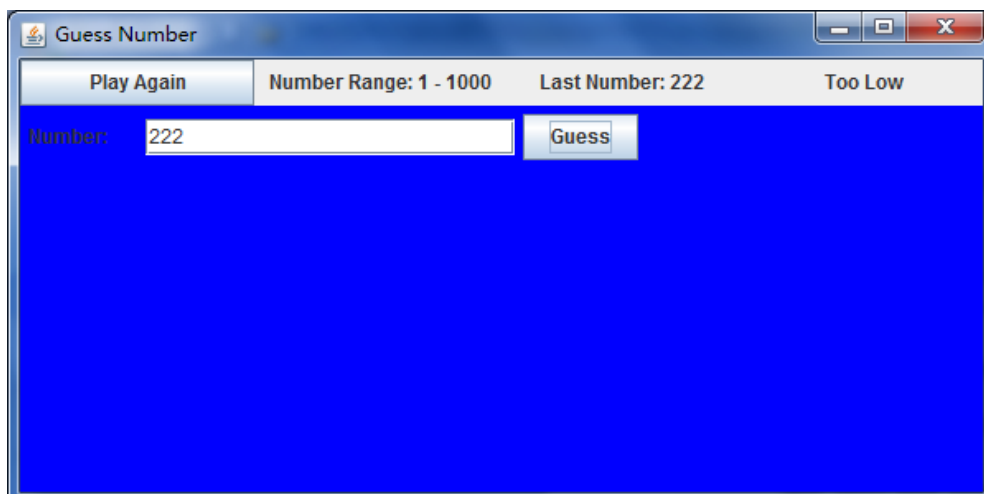
Input an invalid number



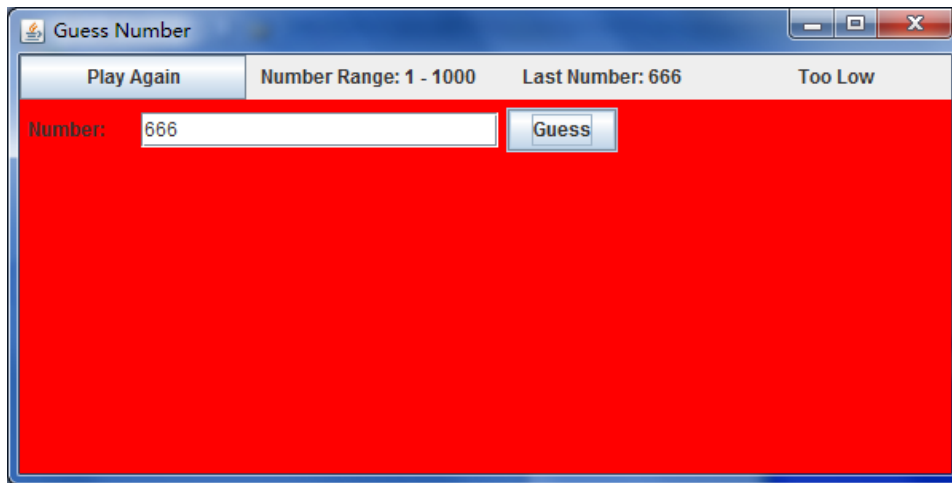
Input a valid number, the color of background changed and we also get the result: "Too low"



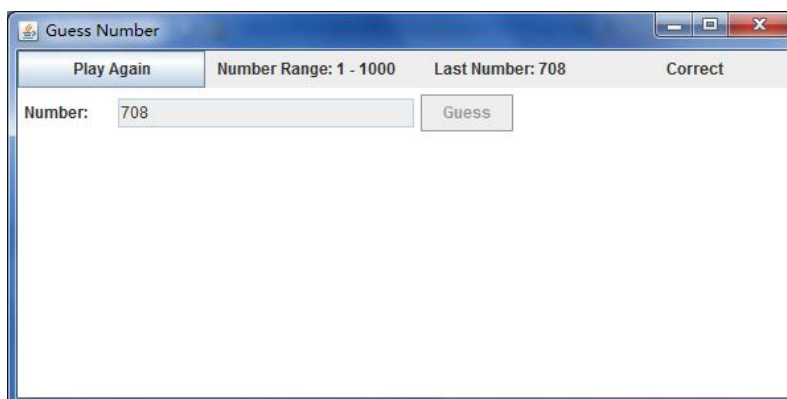
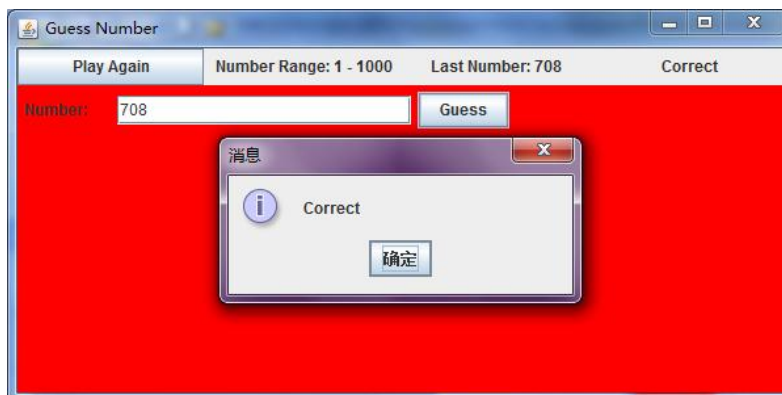
If we guess a smaller number, it will even stray from the correct answer.



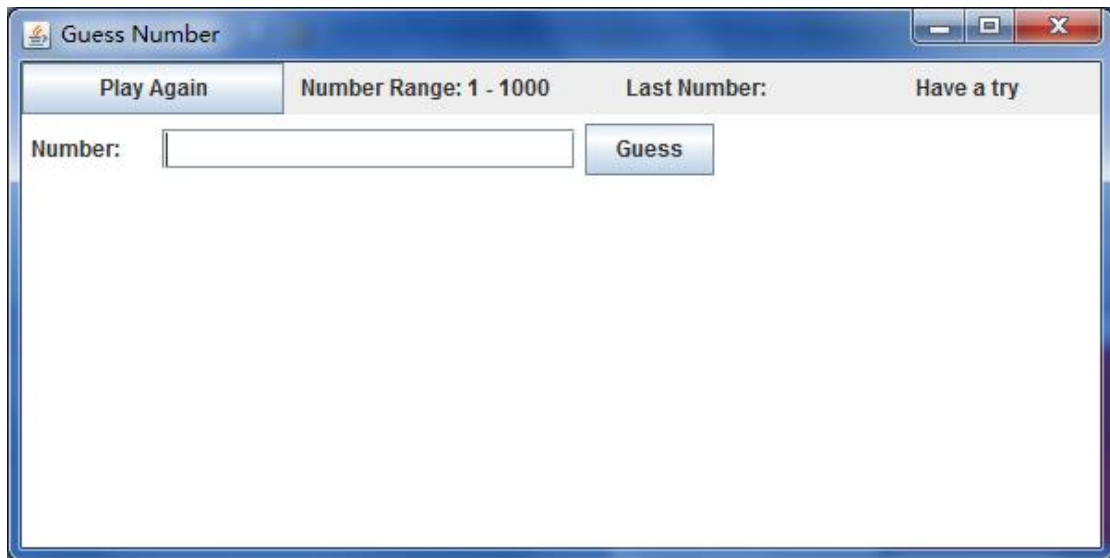
If we guess a bigger number, it will closer to the correct answer.



Keep guessing and we get the correct answer and after clicking the button, the status of the input field and guessing button will change



Clicking play again, it will restart



3. Check Protection

The task is asking us to implement a check protection tool. We need get an input from user and convert it into a safe check amount form.

- a. Use a while loop to get user input and give safe check amount until user enter a negative number to terminate

Note that, the biggest integer digit that a user can input is 6, since we need add commas and decimal into user input.

We deny all invalid input and give a prompt.

```
public static void main(String[] args) {  
    // TODO Auto-generated method stub  
    Scanner in = new Scanner(System.in);  
    System.out.println("Please input the amount:");  
    System.out.println("Enter non-positive number to exit:");  
    double amount = 0;  
    while(true){  
        try{  
            amount = in.nextDouble();  
            if (amount <= 0) {  
                System.out.println("GoodBye!");  
                break;  
            }  
            if (amount >= 1000000) System.out.println("The amount is too big to convert");  
            else printResult(amount);  
        }  
        catch(InputMismatchException e){  
            System.out.println("The amount should be a positive number");  
            in.next();//discard mismatched data  
        }  
        catch(Exception e){// do nothing  
        }  
    }  
    in.close();  
}
```

- b. To parse the user input, we separate amount into two parts, one is integer part, and the other is decimal part.

If user input contains a decimal, use its own decimal; otherwise use 00 as decimal.

```
private static void printResult(double amount) {
    // TODO Auto-generated method stub
    String[] r = Double.toString(amount).split(".");
    char[] i = r[0].toCharArray();
    char[] d = amount == (int)amount? new char[]{'0', '0'} : r[1].toCharArray();
    char[] res = new char[9];
```

- c. Dealing with decimal, just copy decimal to the result array.

Decimal cannot has digit more than 7, since the integer part and "." both at least take one position.

```
int p = 8; // pointer of res
if (d != null) {
    // if has decimal, deal with decimal first
    if (d.length > 7) {
        System.out.println("The amount is too big to convert");
        return;
    }
    else {
        System.arraycopy(d, 0, res, 8 + 1 - d.length, d.length);
        p = 8 - d.length;
        res[p--] = '.';
    }
}
```

- d. Dealing with the integer part, every 4 digit, we add a comma into integer part. If the user input has used up, we complement with "*" and do not add comma. If at the end, the user input still has some digits that are not used; the user input is too big to convert.

```
// deal with integer part
int j = i.length - 1;
int f = 0;
for (; p >= 0; p--) {
    f = (f + 1) % 4;
    if (f == 0 && j != -1) res[p] = ',';
    else {
        res[p] = j >= 0 ? i[j--] : '*';
    }
}
if (j != -1) {
    System.out.println("The amount is too big to convert");
    return;
}
else System.out.println(new String(res));
}
```

e. Final running result

```
CheckProtection [Java Application] C:\Program Files\Ja
Please input the amount:
Enter non-positive number to exit:
123.45
***123.45
1234567
The amount is too big to convert
12345
12,345.00
123.456
**123.456
ccc
The amount should be a positive number
123,44
```


4. Count Duplicate Words

This task is asking us to write a program to count and print the duplicate word in a sentence.

- a. First we need to get sentence from user input, we can make a loop to allow the user keep using our tool until entering an indicator “exit” to terminate.

```
public static void main(String[] args) {  
    // TODO Auto-generated method stub  
    Scanner in = new Scanner(System.in);  
    System.out.println("Please input the sentence:");  
    System.out.println("Enter \"exit\" to exit:");  
    String sentence;  
    while(true){  
        sentence = in.nextLine();  
        if (sentence == null || sentence.length() == 0) continue;  
        if (sentence.equalsIgnoreCase("exit")) {  
            System.out.println("GoodBye!");  
            break;  
        }  
    }  
}
```

- b. Then we pass the sentence to our tool method “count”.

This method will return a Map in which the key is the word (in lower case); the value is the count of the word.

After getting the result, we just traverse the result and output all the duplicate word (count > 1) and their count;

If we don't have duplicate, just print "No duplicate word"

```
Map<String, Integer> res = count(sentence);  
boolean hasDuplicate = false;  
for (String word : res.keySet()){  
    int count = res.get(word);  
    if (count > 1) {  
        hasDuplicate = true;  
        System.out.println(word + ": " + count);  
    }  
}  
if (!hasDuplicate) System.out.println("No Duplicate Word!");  
in.close();  
}
```

- c. For count method, we use a hash Map to maintain the word and corresponding count, so that we can check whether there already exist a certain word efficiently.

To ignore the lowercase and uppercase, we just convert the whole string to lowercase.

To ignore punctuation, we treat all the character that is not a lowercase letter as an end-indicator of a word.

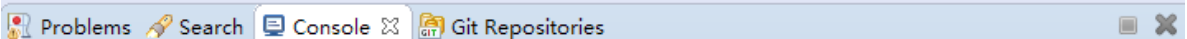
Use string Builder to make the appending of character more efficient.

Whole process: if current character is a lowercase letter, append it to the word and at this time, if the current character is not the last character of the sentence, the word doesn't complete, just keep going.

If reach the end of the sentence or encounter a terminate character, create a new word and update the map. Note that we need ignore all the empty word.

```
public static Map<String, Integer> count(String sentence){
    Map<String, Integer> res = new HashMap<String, Integer>();
    StringBuilder sb = new StringBuilder();
    char[] c = sentence.toLowerCase().toCharArray();
    for (int i = 0; i < c.length; i++){
        if (c[i] >= 'a' && c[i] <= 'z') {
            sb.append(c[i]);
            if (i != c.length - 1) continue; // do not reach the end of the word
        }
        // we get a new word either c[i] is not a character or reach the end of the :
        if (sb.length() == 0) continue; // ignore empty word
        else {
            // get a new word
            String word = sb.toString();
            Integer count = res.get(word);
            if (count == null) res.put(word, 1);
            else res.put(word, count + 1);
            sb = new StringBuilder();
        }
    }
    return res;
}
```

d. Final running result



```
<terminated> CountDuplicateWords [Java Application] C:\Program Files\Java\jdk1.8.0_60\bin\javaw.exe (2016年3月24日 下午4:00)
Please input the sentence:
Enter "exit" to exit:
(No duplicate)catcat dogdog cctt haha
No Duplicate Word!
(one duplicate)cat is a cat
cat: 2
(more duplicates and has punctuation)cat is not a dog;and dog is not a cat,neither!
a: 2
not: 2
and: 2
cat: 2
is: 2
dog: 2
(have empty word)      empty;;;empty      word word
word: 3
empty: 3
(lowercase and UpperCase) lowerCase and upperCase
uppercase: 2
lowercase: 2
and: 2
exit
GoodBye!
```