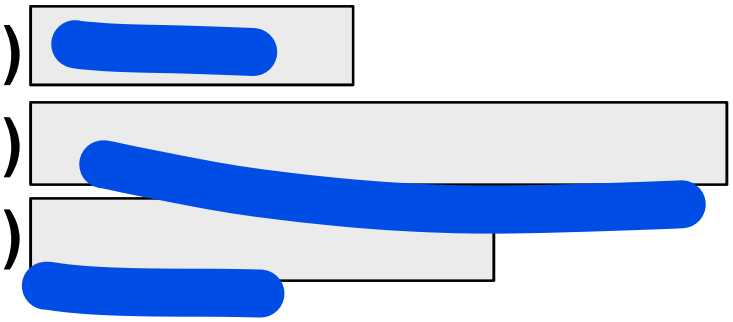
4.2 Programming with loops

- Outline
  - » The Loop Body
  - » Initializing Statements
  - » Ending a Loop
  - » Loop Bugs
  - » Tracing Variables

4.2 Programming with loops

- Three part of a loop
  - » 1)
  - » 2)
  - » 3)



- Techniques for designing each of these loop components

The Loop Body

- To design the loop body, write out the actions the code must accomplish.
- Then look for a [redacted] pattern.
  - » The pattern need not start with the first action.
  - » The repeated pattern will form the body of the loop.
  - » Some actions may need to be done after the pattern stops repeating.

The Loop Body

- The Loop Body : the repeated pattern

Output instructions to the user.
initialize variables
Do the following for the appropriate number of times
{
Read a number into the variable next // repeated pattern.
sum = sum + next;
Output the number and the sum so far
}

Statements

- Some variables need to have a value before the loop begins.
  - » Sometimes this is determined by what is supposed to happen after one loop iteration.
  - » Often variables have an initial value of zero or one, but not always.
- Other variables get values only while the loop is iterating.

Initializing Statements

- sum must be initialized to 0;

```
Output instructions to the user.
sum = 0;
Do the following for the appropriate number of times
{
        Read a number into the varialbe next  // repeated pattern.
                sum = sum + next;
        Output the number and the sum so far
}
```

Initializing Statements

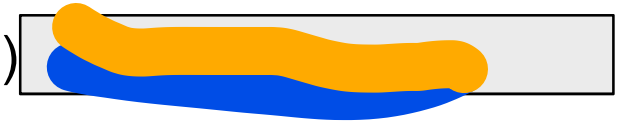
- Variables are not always initialized to Zero

```
for (count = 1; count <= n; count ++)
{
        Read a number into the varialbe next  // repeated pattern.
        product = product * count;
}
```

Ending a Loop

- If the number of iterations is known before the loop starts, the loop is called a [redacted] *loop.*
  - » use a [redacted] loop.
- Asking the user before each iteration if it is time to end the loop is called the [redacted] *technique.*
  - » appropriate for a small number of iterations
  - » Use a [redacted] loop or a [redacted] loop

Ending a Loop

- 1)  -
- 2)  –
- 3) 
  - » used to signal the end of the input
  - »

Count-Controlled Loops

```
double next, average, sum = 0;
for (int count = 1; count <= numberOfStudents; count++)
{
     next = keyboard.nextDouble();
     sum = sum + next;
}
if (numberOfStudents > 0)
     average = sum/numberOfStudents;
else
     System.out.println("No scores to average.");
```

ask – before – iterating

```
do
{
    System.out.println("Enter price $");
    price = keyboard.nextDouble();
    System.out.print("Enter number purchased:");
    number = keyboard.nextInt();
    System.out.println(number + " items at $" + price);
    System.out.println("Total cost $" + price * number);
    System.out.println("Want to make another purchase?");
    System.out.println("Enter yes or no.");
    answer = keyboard.next();
} while (answer.equalsIgnoreCase("yes"));
```

```
import java.util.*;
public class ExamAverager
{
   public static void main(String[] args)
   {
……..
……..
        double next;
        String answer;
        Scanner keyboard = new Scanner(System.in);
        do
        {
……..
……..
……..
            System.out.println("Want to average another exam?");
            System.out.println("Enter yes or no.");
            answer = keyboard.next( );
        }while (answer.equalsIgnoreCase("yes"));

   }
}
```

sentinel value

- For large input lists, a *sentinel value* can be used to signal [redacted] of the list.
  - » The sentinel value must be [redacted] from all the other possible inputs.
  - » Ex) A negative number following a long list of nonnegative exam scores could be suitable.

```
90
0
10
-1
```

sentinel value

```
System.out.println("Enter scores for all students.");
System.out.println("Enter a negative number after");
System.out.println("you have entered all the scores.");
Scanner keyboard = new Scanner(System.in);
double max = keyboard.nextDouble();
double min = max; //The max and min so far are the first score.
double next = keyboard.nextDouble();
while (next >= 0)
{
    if (next > max)
        max = next;
    else if (next < min)
        min = next;
    next = keyboard.nextDouble();
}
System.out.println("The highest score is " + max);
System.out.println("The lowest score is " + min);
```

```
100
90
10
-1
```

The output will be

```
The highest score is 100
The lowest score is 10
```

- example - reading a list of scores followed by a sentinel value

```
int next = keyboard.nextInt();
while (next  >= 0)
{
   Process_The_Score
   next = keyboard.nextInt();
}
```

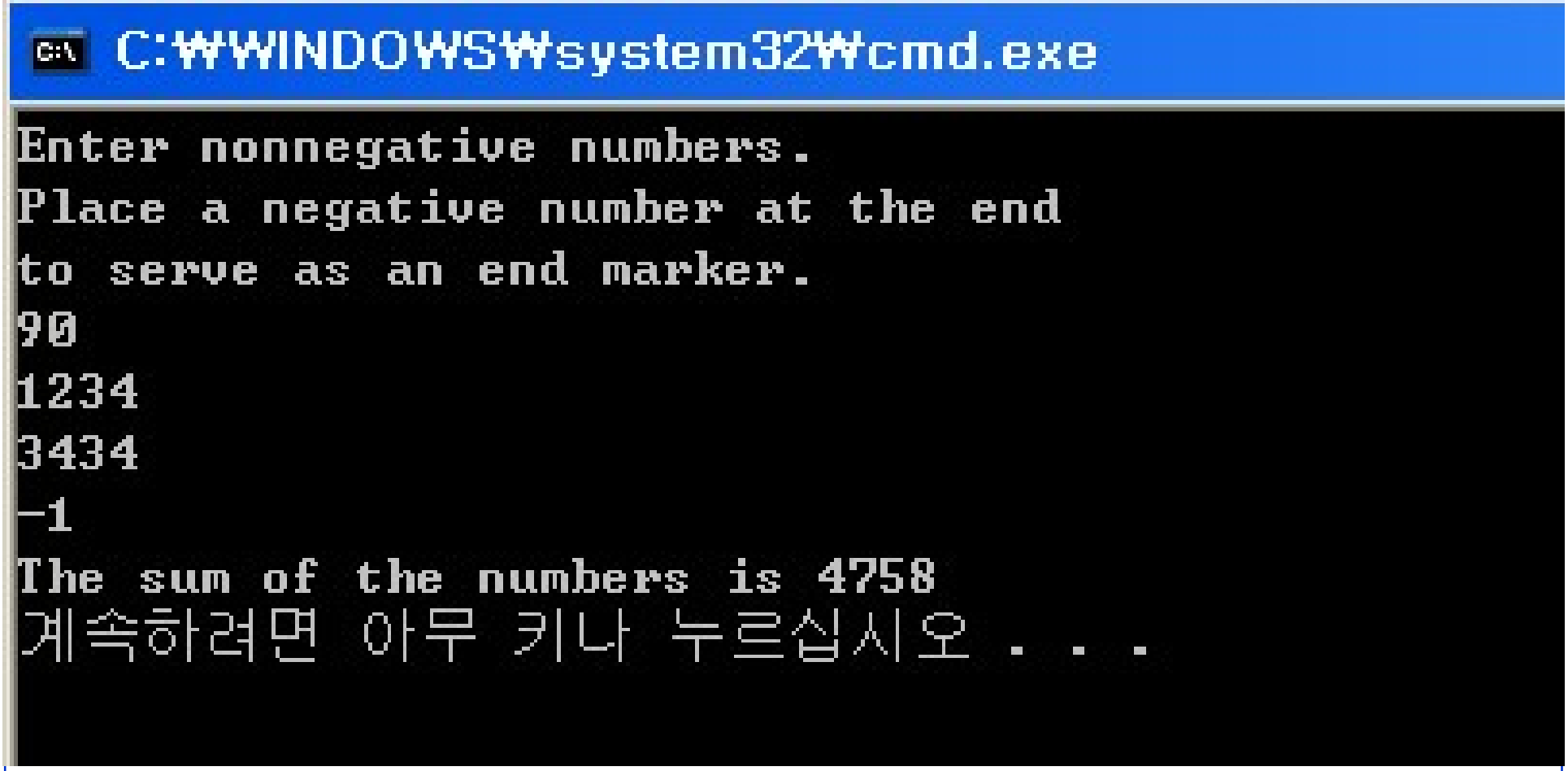
Declaring Variables Outside Loop Bodies

- The declaration of variables **inside** a loop body is repeated with each execution of the loop body.
  - » This can be inefficient, depending on the compiler.
- It the declaration of variables can be moved **outside** the loop body, generally it is appropriate to do so.

```
// Listing 4.6

import java.util.Scanner;
/**
Illustrates the use of a boolean variable to end loop iteration.
*/
public class BooleanDemo
{
    public static void main (String [] args)
    {
        System.out.println ("Enter nonnegative numbers.");
        System.out.println ("Place a negative number at the end");
        System.out.println ("to serve as an end marker.");
        int sum = 0;
        boolean areMore = true;
        Scanner keyboard = new Scanner (System.in);
```

```
        while (areMore)
        {
            int next = keyboard.nextInt ();
            if (next < 0)
                areMore = false;
            else
                sum = sum + next;
        }
        System.out.println ("The sum of the numbers is " + sum);
    }
}
```

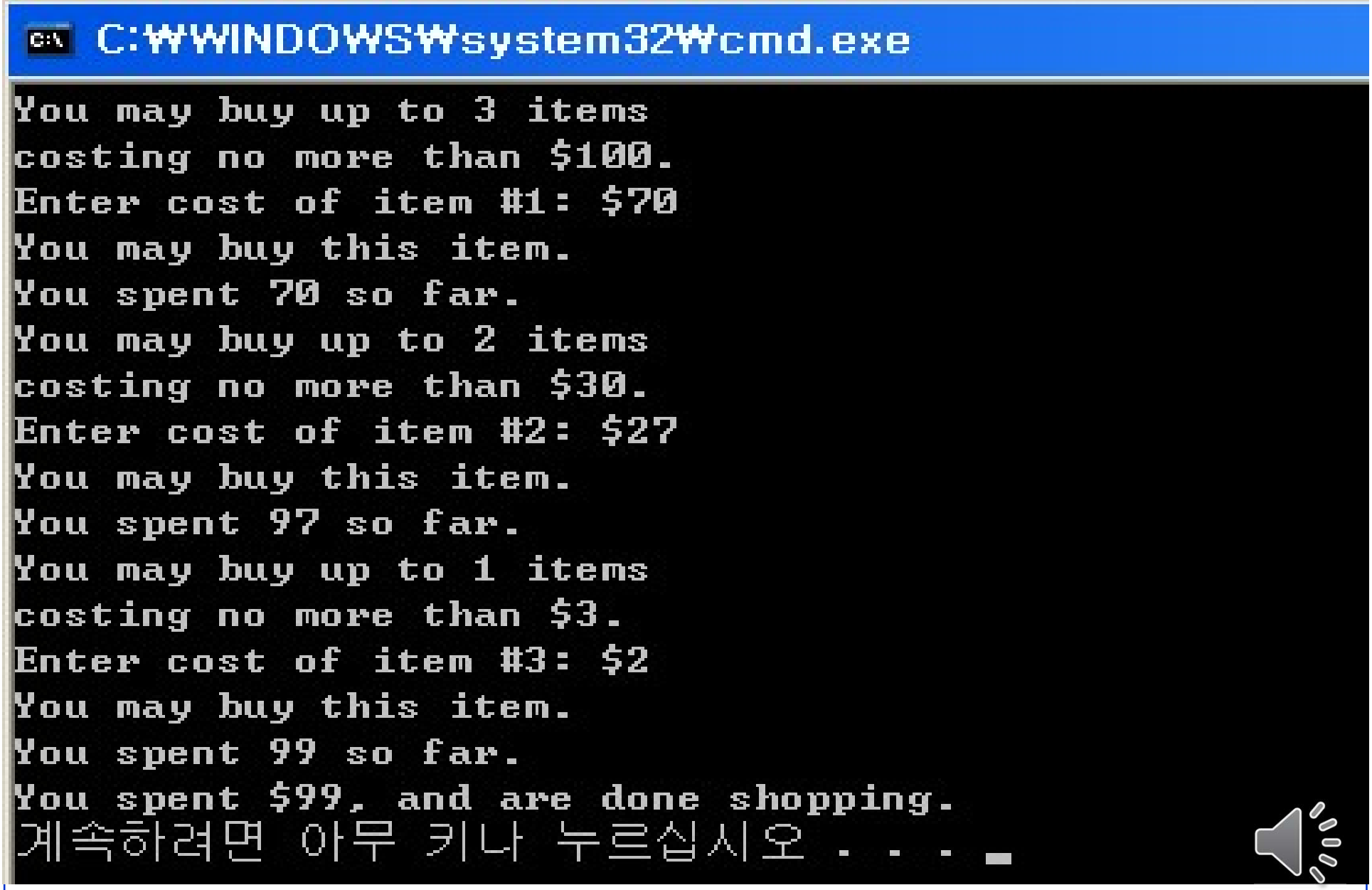

C:₩WINDOWS₩system32₩cmd.exe
Enter nonnegative numbers.
Place a negative number at the end
to serve as an end marker.
90
1234
3434
-1
The sum of the numbers is 4758
계속하려면 아무 키나 누르십시오 . . .

// Listing 4.7

```
import java.util.Scanner;
public class SpendingSpree
{
    public static final int SPENDING_MONEY = 100;
    public static final int MAX_ITEMS = 3;
    public static void main (String [] args)
    {
        Scanner keyboard = new Scanner (System.in);
        boolean haveMoney = true;
        int leftToSpend = SPENDING_MONEY;
        int totalSpent = 0;
        int itemNumber = 1;
        while (haveMoney && (itemNumber <= MAX_ITEMS))
        {
```

```
System.out.println ("You may buy up to " +
        (MAX_ITEMS - itemNumber + 1) +
        " items");
System.out.println ("costing no more than $" +
        leftToSpend + ".");
System.out.print ("Enter cost of item #" +
        itemNumber + ": $");
int itemCost = keyboard.nextInt ();
if (itemCost <= leftToSpend)
{
    System.out.println ("You may buy this item. ");
    totalSpent = totalSpent + itemCost;
    System.out.println ("You spent " + totalSpent + " so far. ");
            leftToSpend = SPENDING_MONEY - totalSpent;
```

```
            if (leftToSpend > 0)
                itemNumber++;
            else
            {
                System.out.println ("You are out of money.");
                    haveMoney = false;
            }
        }
        else
            System.out.println ("You cannot buy that item.");
    }
    System.out.println ("You spent $" + totalSpent +
            ", and are done shopping.");
  }
}
```


C:₩WINDOWS₩system32₩cmd.exe
You may buy up to 3 items
costing no more than $100.
Enter cost of item #1: $70
You may buy this item.
You spent 70 so far.
You may buy up to 2 items
costing no more than $30.
Enter cost of item #2: $27
You may buy this item.
You spent 97 so far.
You may buy up to 1 items
costing no more than $3.
Enter cost of item #3: $2
You may buy this item.
You spent 99 so far.
You spent $99, and are done shopping.
계속하려면 아무 키나 누르십시오 . . .

Loop Bugs

- common loop bugs
  - » unintended infinite loops
  - » off-by-one errors
  - » testing equality of floating-point numbers
- subtle infinite loops
  - » The loop may terminate for some input values, but not for others.
  - » For example, you can't get out of debt when the monthly penalty exceeds the monthly payment.

Some Practical Considerations When Using Loops

- The most common loop errors are unintended infinite loops and off-by-one errors in counting loops.
- Sooner or later *everyone* writes an unintentional infinite loop
  - » To get out of an unintended infinite loop enter [redacted]
  ([redacted])

Subtle Infinite Loops

- Verify that the monthly payment exceeds the penalty, for example, before entering a loop to determine the number of payments needed to get out of debt.

```
if (payment <= penalty)
   System.out.println("payment is too
   small");
else
{
  ...
```

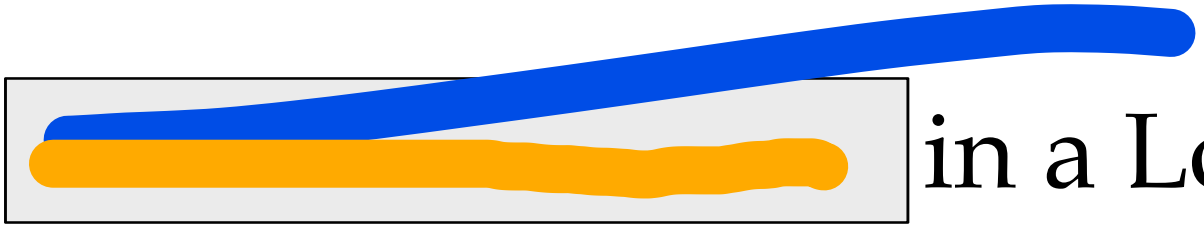
Off-by-One Errors

- The loop body is repeated one too many times or one too few times.
- examples
  - » < is used when <= should be used or <= is used when < should be used
  - » using the index of the last character of a string instead of the length of the string (or vice versa)
- easy to overlook

Testing Equality of Floating-point Numbers

- == works satisfactorily for [redacted] and [redacted].
- == is not reliable for floating-point numbers (which are approximate quantities).
  - » Use <= or >= rather than == or !=.

- Loops should be tested thoroughly, especially at the boundaries of the loop test, to check for off-by-one and other possible errors.
- [redacted] error : loop repeats the loop body one to many times or one too few times.
- Use of == to test for equality
  - » satisfactorily for integers and characters
  - » but not reliable for [redacted] numbers.
  - » ➔ … <=, do not use ==, !=



in a Loop

- *a variable*: print out the variable each time through the loop
- A common technique is to test loop counters and troubleshoot off-by-one and other loop errors.
- Some systems provide a built-in tracing system that allows you to trace a variable without having to change your program.
- If no built-in utility is available, insert temporary output statements to print values.

- *Tracing variables* means watching the variables change while the program is running.
  - » Simply insert temporary output statements in your program to print of the values of variables of interest
  - » or, learn to use the debugging facility that may be provided by your system.

```
int time;
for (time = 1; time <= 4; time++)
    System.out.println("One more time.");


int result = 1;
int count;
for (count = 1; count <= 5; count++)
    result = 2*result;
```

```
0
0
1
0
1
2
```

```
count=0;
while (balance <0)
{
        balance = balance + penalty;
        balance = balance + deposit;
        count++;
}
System.out.println("Nonnegative balance in " +
                count + " months.");
```

```
count=0;
System.out.println("count == " + count); //trace
System.out.println("balance== " + balance); //trace
System.out.println("penalty == " + penalty); //trace
System.out.println("deposit == " + deposit); //trace

while (balance <0)
{
        balance = balance + penalty;
        System.out.println(
                "balance+panalty == " + balance); //trace

        balance = balance - deposit;
        System.out.println(
                "balance-panalty == " + balance); //trace

        count++;
        System.out.println("count == " + count); //trace

}
System.out.println("Nonnegative balance in " +
                count + " months.");
```