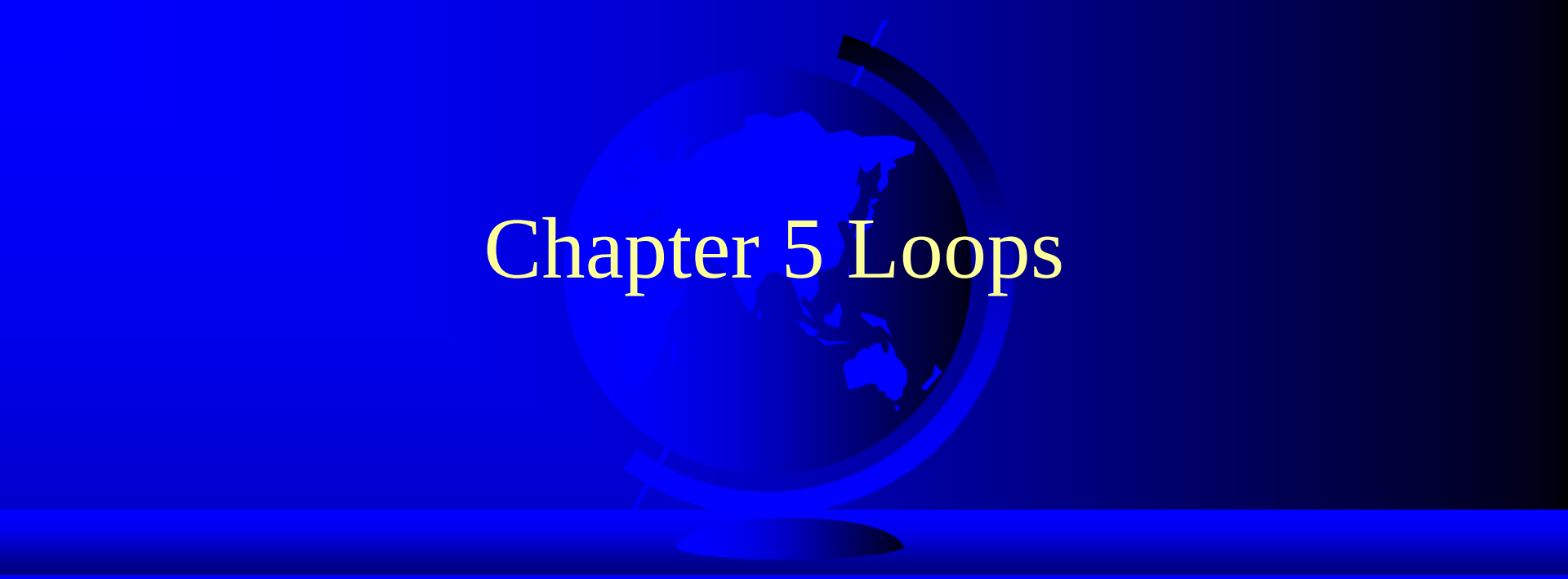


Chapter 5 Loops

Objectives

To write programs for executing statements repeatedly using a while loop

To develop a program for GuessNumber and SubtractionQuizLoop

To follow the loop design strategy to develop loops

To develop a program for SubtractionQuizLoop

To control a loop with a sentinel value

To obtain large input from a file using input redirection rather than typing from the keyboard

To write loops using do-while statements

To write loops using for statements

To discover the similarities and differences of three types of loop statements

To write nested loops

To learn the techniques for minimizing numerical errors

To learn loops from a variety of examples (GCD, FutureTuition, MonteCarloSimulation)

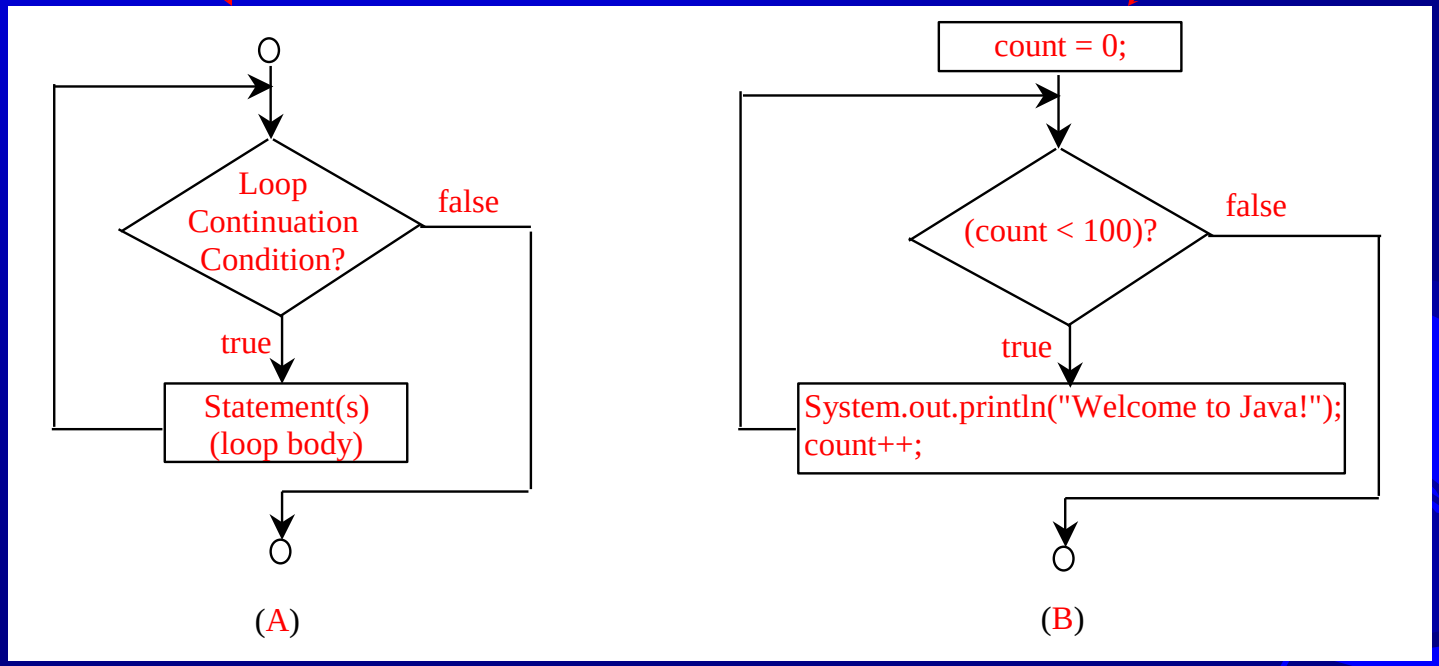
To implement program control with break and continue

(GUI) To control a loop with a confirmation dialog

`while` Loop Flow Chart

```
while (loop-continuation-condition) {
  // loop-body;
  Statement(s);
}
```

```
int count = 0;
while (count < 100) {
  System.out.println("Welcome to Java!");
  count++;
}
```



animation

Trace while Loop

Initialize count

```
int count = 0;
while (count < 2) {
  System.out.println("Welcome to Java!");
  count++;
}
```

animation

Trace while Loop, cont.

(count < 2) is true

```
int count = 0;

while (count < 2) {

  System.out.println("Welcome to Java!");

  count++;

}
```

animation

Trace while Loop, cont.

```
int count = 0;
while (count < 2) {
  System.out.println("Welcome to Java!");
  count++;
}
```

Print Welcome to Java

animation

Trace while Loop, cont.

```
int count = 0;
while (count < 2) {
  System.out.println("Welcome to Java!");
  count++;
}
```

Increase count by 1
count is 1 now

animation

Trace while Loop, cont.

```
int count = 0;
while (count < 2) {
  System.out.println("Welcome to Java!");
  count++;
}
```

(count < 2) is still true since count is 1

animation

Trace while Loop, cont.

```
int count = 0;
while (count < 2) {
  System.out.println("Welcome to Java!");
  count++;
}
```

Print Welcome to Java

animation

Trace while Loop, cont.

```
int count = 0;
while (count < 2) {
  System.out.println("Welcome to Java!");
  count++;
}
```

Increase count by 1
count is 2 now

animation

Trace while Loop, cont.

```
int count = 0;
while (count < 2) {
  System.out.println("Welcome to Java!");
  count++;
}
```

(count < 2) is false since count is 2 now

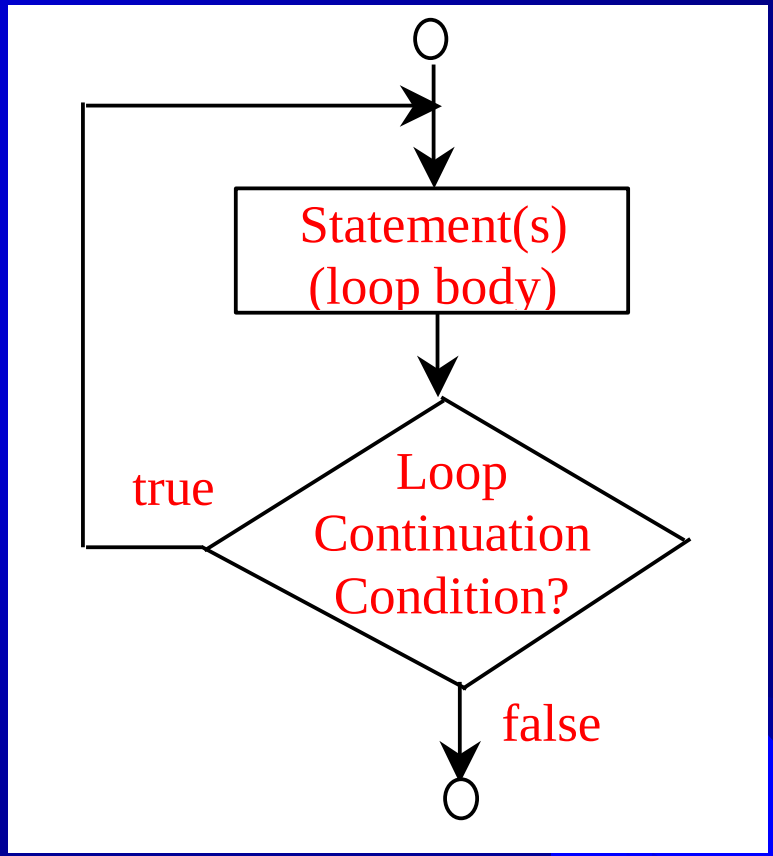
animation

Trace while Loop

```
int count = 0;
while (count < 2) {
  System.out.println("Welcome to Java!");
  count++;
}
```

The loop exits. Execute the next statement after the loop.

do-while Loop

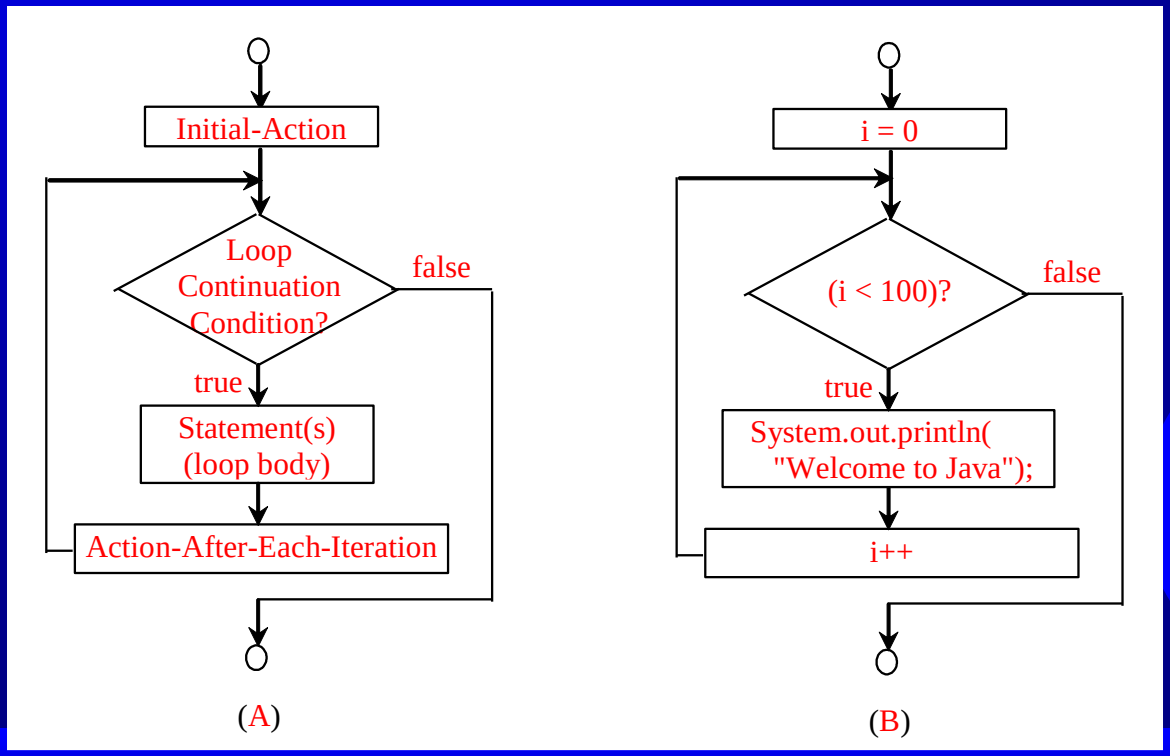



```
do {
  // Loop body;
  Statement(s);
} while (loop-continuation-condition);
```

for Loops

```
for (initial-action; loop-
    continuation-condition;
    action-after-each-iteration) {
  // loop body;
  Statement(s);
}
```

```
int i;
for (i = 0; i < 100; i++) {
  System.out.println(
    "Welcome to Java!");
}
```



animation

Trace for Loop

Declare i

```
int i;
for (i = 0; i < 2; i++) {
  System.out.println(
    "Welcome to Java!");
}
```

animation

Trace for Loop, cont.

```
int i;
for (i = 0; i < 2; i++) {
  System.out.println(
    "Welcome to Java!");
}
```

Execute initializer
i is now 0

animation

Trace for Loop, cont.

```
int i;
for (i = 0; i < 2; i++) {
  System.out.println( "Welcome to Java!");
}
```

(i < 2) is true
since i is 0

animation

Trace for Loop, cont.

```
int i;
for (i = 0; i < 2; i++) {
  System.out.println("Welcome to Java!");
}
```

Print Welcome to Java

animation

Trace for Loop, cont.

```
int i;
for (i = 0; i < 2; i++) {
  System.out.println("Welcome to Java!");
}
```

Execute adjustment statement
i now is 1

animation

Trace for Loop, cont.

```
int i;
for (i = 0; i < 2; i++) {
  System.out.println("Welcome to Java!");
}
```

(i < 2) is still true
since i is 1

animation

Trace for Loop, cont.

```
int i;
for (i = 0; i < 2; i++) {
  System.out.println("Welcome to Java!");
}
```

Print Welcome to Java

animation

Trace for Loop, cont.

```
int i;
for (i = 0; i < 2; i++) {
  System.out.println("Welcome to Java!");
}
```

Execute adjustment statement
i now is 2

animation

Trace for Loop, cont.

```
int i;
for (i = 0; i < 2; i++) {
  System.out.println("Welcome to Java!");
}
```

(i < 2) is false
since i is 2

animation

Trace for Loop, cont.

```
int i;
for (i = 0; i < 2; i++) {
  System.out.println("Welcome to Java");
}
```

Exit the loop. Execute the next statement after the loop

Note

The initial-action in a for loop can be a list of zero or more comma-separated expressions.

The action-after-each-iteration in a for loop can be a list of zero or more comma-separated statements.

Therefore, the following two for loops are correct, but rarely used in practice.

```
for (int i = 0, j = 0; (i + j < 10); i++, j++) {
    // Do something
}

for (int i = 1; i < 100; System.out.println(i++) );
```

Note

If the loop-continuation-condition in a for loop is omitted, it is implicitly true.

Thus the statement given below in (a), which is an infinite loop, is correct.

Nevertheless, it is better to use the equivalent loop in (b) to avoid confusion:

```
for ( ; ; ) {
  // Do something
}
```

(a)

Equivalent

```
while (true) {
  // Do something
}
```

(b)

Caution

Adding a semicolon at the end of the for clause before the loop body is a common mistake, as shown below:

Logic Error

```
for (int i=0; i<10; i++);
{
  System.out.println("i is " + i);
}
```

Caution, cont.

Similarly, the following loop is also wrong:

```
int i=0;
while (i < 10);
{
  System.out.println("i is " + i);
  i++;
}
```

Logic Error

In the case of the do loop, the following semicolon is needed to end the loop.

```
int i=0;
do {
  System.out.println("i is " + i);
  i++;
} while (i<10);
```

Correct

Which Loop to Use?

The three forms of loop statements, while, do-while, and for, are expressively equivalent; that is, you can write a loop in any of these three forms.

For example, a while loop in (a) in the following figure can always be converted into the following for loop in (b):

```
while (loop-continuation-condition) {
  // Loop body
}
```

(a)

Equivalent

```
for ( ; loop-continuation-condition; )
  // Loop body
}
```

(b)

can generally be converted into the following while loop in (b) except in certain special cases (see Review Question 4.17 for one of them):

```
for (initial-action;
     loop-continuation-condition;
     action-after-each-iteration) {
  // Loop body;
}
```

(a)

Equivalent

```
initial-action;
while (loop-continuation-condition) {
  // Loop body;
  action-after-each-iteration;
}
```

(b)

The `for` loop on the left is converted into the `while` loop on the right. What is wrong? Correct it.

```
for (int i = 0; i < 4; i++) {
  if (i % 3 == 0) continue;
  sum += i;
}
```

Converted

Wrong conversion

```
int i = 0;
while (i < 4) {
  if(i % 3 == 0) continue;
  sum += i;
  i++;
}
```

If a continue statement is executed inside a loop, the rest of the iteration is skipped.

Solution

If a continue statement is executed inside a for loop, the rest of the iteration is skipped, then the action-after-each-iteration is performed and the loop-continuation-condition is checked.

If a continue statement is executed inside a while loop, the rest of the iteration is skipped, then the loop-continuation-condition is checked.

```
for (int i = 0; i < 4; i++) {
  if (i % 3 == 0) continue;
  sum += i;
}
```

```
int i = 0;

while (i < 4) {
  if (i % 3 == 0) {
    i++;
    continue;
  }
  sum += i;
  i++;
}
```

Algorithm of Success

```
while(noSuccess)
{
        tryAgain();

        if(Dead)
                break;
}
```

Using `break` and `continue`

You have used the keyword **break** in a **switch** statement.
You can also use **break** in a loop to immediately terminate the loop.

TestBreak.java

```
1 public class TestBreak {
2   public static void main(String[] args) {
3     int sum = 0;
4     int number = 0;
5
6     while (number < 20) {
7       number++;
8       sum += number;
9       if (sum >= 100)
10        break;
11    }
12
13    System.out.println("The number is " + number);
14    System.out.println("The sum is " + sum);
15  }
16 }
```

```
The number is 14
The sum is 105
```

Using `break` and `continue`

break ends the whole loop.

continue ends the current iteration

so that the rest of the statement in the loop body is not executed

TestContinue.java

```
1 public class TestContinue {
2   public static void main(String[] args) {
3     int sum = 0;
4     int number = 0;
5
6     while (number < 20) {
7       number++;
8       if (number == 10 || number == 11)
9         continue;
10      sum += number;
11    }
12
13    System.out.println("The sum is " + sum);
14  }
15 }
```

```
The sum is 189
```

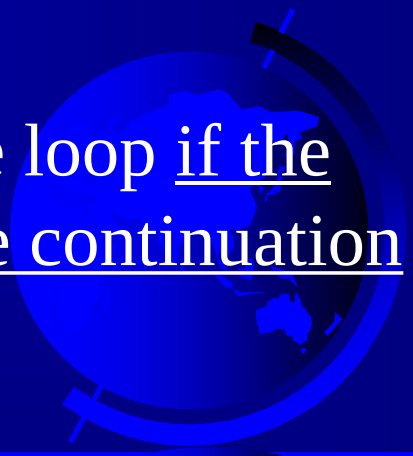
Recommendations

Use the one that is most intuitive and comfortable for you.

In general, a for loop may be used if the number of repetitions is known, as, for example, when you need to print a message 100 times.

A while loop may be used if the number of repetitions is not known, as in the case of reading the numbers until the input is 0.

A do-while loop can be used to replace a while loop if the loop body has to be executed before testing the continuation condition.



Minimizing Numerical Errors

Numeric errors involving floating-point numbers are inevitable.

How to minimize such errors?

use integer (rather than a floating point) count as a control variable

presents an example summing a series that starts with 0.01 and ends with 1.0. The numbers in the series will increment by 0.01, as follows: 0.01 + 0.02 + 0.03 and so on.

TestSum.java

```
1 public class TestSum {
2   public static void main(String[] args) {
3     // Initialize sum
4     float sum = 0;
5
6     // Add 0.01, 0.02, ..., 0.99, 1 to sum
7     for (float i = 0.01f; i <= 1.0f; i = i + 0.01f)
8       sum += i;
9
10    // Display result
11    System.out.println("The sum is " + sum);
12  }
13 }
```

```
The sum is 50.499985
```

But the exact **sum** should be **50.50.**

Because computers use a fixed number of bits to represent floating-point numbers, and thus they cannot represent some floating-point numbers exactly.

How about double (64 bits)?

```
// Initialize sum
double sum = 0;

// Add 0.01, 0.02, ..., 0.99, 1 to sum
for (double i = 0.01; i <= 1.0; i = i + 0.01)
  sum += i;
```

However, you will be stunned to see that the result is actually 49.50000000000003. What went wrong? If you print out i for each iteration in the loop, you will see that the last i is slightly larger than 1 (not exactly 1). This causes the last i not to be added into sum. The fundamental problem is that the floating-point numbers are represented by approximation. To fix

```
double currentValue = 0.01;
for (int count = 0; count < 100; count++) {
  sum += currentValue;
  currentValue += 0.01;
}
```

Caution

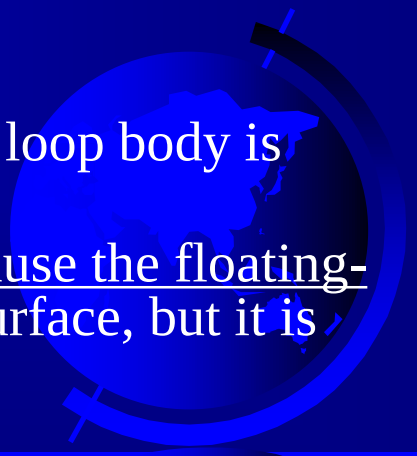
Don't use floating-point values for equality checking in a loop control.

- Since floating-point values are approximations for some values, using them could result in imprecise counter values and inaccurate results.
- Consider the following code for computing 1 + 0.9 + 0.8 + ... + 0.1:

```
double sum = 0;
double item = 1;
while (item != 0) { // No guarantee item will be 0
  sum += item;
  item -= 0.1;
}
System.out.println(sum);
```

Variable item starts with 1 and is reduced by 0.1 every time the loop body is executed. The loop should terminate when item becomes 0.

However, there is no guarantee that item will be exactly 0, because the floating-point arithmetic is approximated. This loop seems OK on the surface, but it is actually an infinite loop.



Input Redirection

This command is called *input redirection*. The program takes the input from the file **input.txt** rather than having the user type the data from the keyboard at runtime. Suppose the contents of the file are as follows:

```
2 3 4 5 6 7 8 9 12 23 32
23 45 67 89 92 12 34 35 3 1 2 4 0
```

The program should get `sum` to be `518`.

- Read input using **input redirection** from a file

Suppose you have created a text file named LottoNumbers.txt that contains the input data `2 5 6 5 4 3 23 43 2 0`. You can run the program using the following command:

```
java LottoNumbers < LottoNumbers.txt
```

Input and Output Redirection

Run program using the **input redirection** command:

java *ClassName* < *input.txt*

- To input a large number of data
- Input from a data text file, say *input.txt*, which stores the data separated by whitespaces

Similarly, **output redirection** sends the output to a file, say *output.txt*, rather than displaying it on the console.

java *ClassName* > *output.txt*

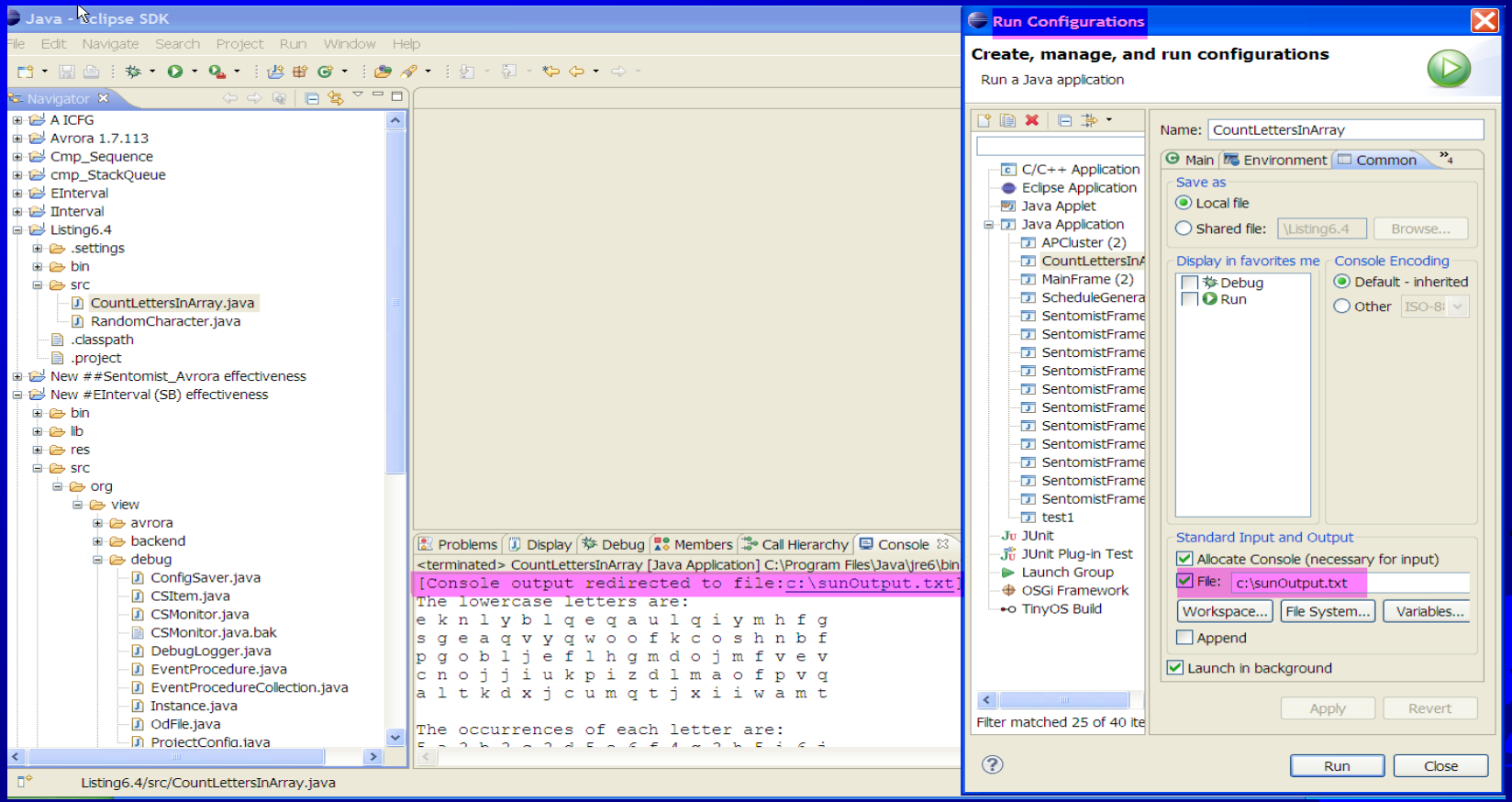
Input and output redirection can be used in the same command. :

java *ClassName* < *input.txt* > *output.txt*

In Eclipse

Run as->Run Configurations -> Common -> Stand Input and Output

Output to a file:



Using *System.setIn()* to change the InputStream

```
public void test() throws Exception {
    System.setIn( new FileInputStream("d:\\temp\\fsd.txt") );
    Scanner sc = new Scanner( System.in );
    System.out.println( sc.nextLine() );
    sc.close();
}
```

Case Studies

Problem: Finding the Greatest Common Divisor

Problem: Write a program that prompts the user to enter two positive integers and finds their greatest common divisor.

(4, 2) : ? (16, 24): ?

```
Enter first integer: 125 Enter
Enter second integer: 2525 Enter
The greatest common divisor for 125 and 2525 is 25
```

Let the two input integers be n1 and n2.

- 1 is a common divisor, but may not be the greatest one
- So you can check whether k (for k = 2, 3, 4, and so on) is a common divisor for n1 and n2, until k is greater than n1 or n2.

GreatestCommonDivisor.java

```
 1 import java.util.Scanner;
 2
 3 public class GreatestCommonDivisor {
 4   /** Main method */
 5   public static void main(String[] args) {
 6     // Create a Scanner
 7     Scanner input = new Scanner(System.in);
 8
 9     // Prompt the user to enter two integers
10     System.out.print("Enter first integer: ");
11     int n1 = input.nextInt();
12     System.out.print("Enter second integer: ");
13     int n2 = input.nextInt();
14
15     int gcd = 1; // Initial gcd is 1
16     int k = 2; // Possible gcd
17     while (k <= n1 && k <= n2) {
18       if (n1 % k == 0 && n2 % k == 0)
19         gcd = k; // Update gcd
20       k++;
21     }
22
23     System.out.println("The greatest common divisor for " + n1 +
24       " and " + n2 + " is " + gcd);
25   }
26 }
```

Problem: Displaying Prime Numbers

An integer greater than 1 is ***prime,*** if its only positive divisor is 1 or itself.

- 2, 3, 4, 5, 6, 7, 8, 9, …?

Problem: Write a program that displays the first 50 prime numbers in five lines, each of which contains 10 numbers.

Solution: The problem can be broken into the following tasks:

- For number = 2, 3, 4, 5, 6, ..., test whether the number is prime.
- Determine whether a given number is prime.
- Count the prime numbers.
- Print each prime number, and print 10 numbers per line.

Algorithm: Write a loop and repeatedly test whether a **number** is prime. If the **number** is prime, increase the count by **1**.

The **count** is **0** initially. When it reaches **50**, the loop terminates.

```
Set the number of prime numbers to be printed as
  a constant NUMBER_OF_PRIMES;
Use count to track the number of prime numbers and
  set an initial count to 0;
Set an initial number to 2;

while (count < NUMBER_OF_PRIMES) {
  Test whether number is prime;

  if number is prime {
    Print the prime number and increase the count;
  }

  Increment number by 1;
}
```

To test whether a number is prime, check whether it is divisible by 2, 3, 4, up to number/2. If a divisor is found, the number is not a prime. The algorithm can be described as follows:

```
Use a boolean variable isPrime to denote whether
  the number is prime; Set isPrime to true initially;

for (int divisor = 2; divisor <= number / 2; divisor++) {
  if (number % divisor == 0) {
    Set isPrime to false
    Exit the loop;
  }
}
```

(GUI) Controlling a Loop with a Confirmation Dialog

The answers *Yes* or *No* to continue or terminate the loop. The template of the loop may look as follows:

```
 7      // Keep reading data until the user answers No
 8      int option = JOptionPane.YES_OPTION;
 9      while (option == JOptionPane.YES_OPTION) {
10        // Read the next data
11        String dataString = JOptionPane.showInputDialog(
12          "Enter an int value: ");
13        int data = Integer.parseInt(dataString);
14
15        sum += data;
16
17        option = JOptionPane.showConfirmDialog(null, "Continue?");
18      }
```