JAVA PRIMER: I/O METHODS AND CONTROL FLOW

1

Simple Output

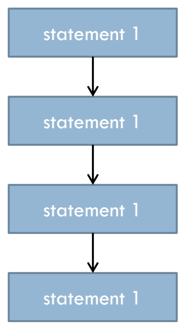
- Java provides a built-in static object, called System.out, that performs output to the "standard output" device, with the following methods:

print(String *s*):	Print the string *s*.
print(Object *o*):	Print the object *o* using its toString method.
print(*baseType* *b*):	Print the base type value *b*.
println(String *s*):	Print the string *s*, followed by the newline character.
println(Object *o*):	Similar to print(*o*), followed by the newline character.
println(*baseType* *b*):	Similar to print(*b*), followed by the newline character.

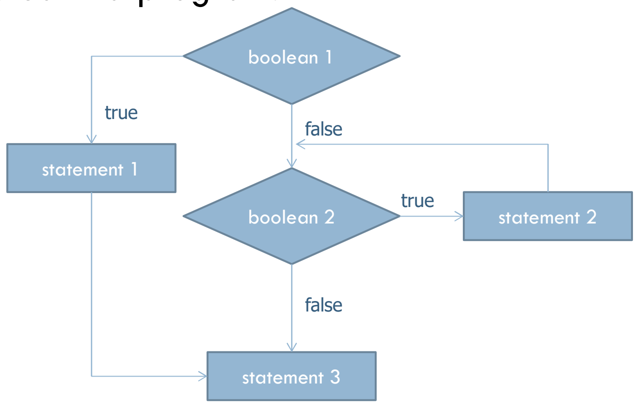
2

Flow Control

- Control flow is the squence of statements that are actually executed in a program.



straight-line control flow



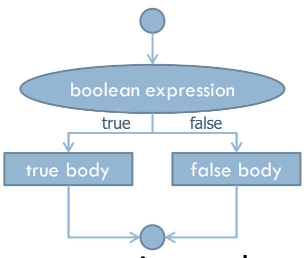
control flow with conditions and loops

3

If Statements

- The syntax of a simple **if** statement is as follows:

if (*booleanExpression*)
    trueBody
else
    falseBody



- booleanExpression is a boolean expression and trueBody and falseBody are each either a single statement or a block of statements enclosed in braces ("**{**" and "**}**").

4

Ex. If Statement

- Heads or Tails

```
public class Flip {
    public static void main(String[] args) {
        if (Math.random() < 0.5)
                System.out.println("Heads");
        else
                System.out.println("Tails");
    }
}
```

```
% java Flip
Heads
% java Flip
Heads
% java Flip
Tails
% java Flip
Heads
```

5

If Statement Examples

Absolute value	`if (x < 0) x = -x;`

Put x and y into sorted order	`if (x>y)`
	`{`
	`    int t = x;`
	`    x = y;`
	`    y = t;`
	`}`

Maximum of x and y	`if (x > y) max = x;`
	`else max = y;`

Error check for division operation	`if (den == 0) System.out.println(“Division by zero”);`
	`else System.out.println(“Quotient = “ + num/den);`

Error check for quadratic formula	`double discriminant = b*b – 4.0*c;`
	`if (discriminant < 0.0)`
	`{`
	`    System.out.println(“No real roots”);`
	`}`
	`else`
	`{`
	`    System.out.println((-b + Math.sqrt(discriminant))/2.0);`
	`    System.out.println((-b - Math.sqrt(discriminant))/2.0);`
	`}`

6

Compound if Statements

- There is also a way to group a number of boolean tests, as follows:

```
if  (firstBooleanExpression)
      firstBody
else if  (secondBooleanExpression)
      secondBody
else
      thirdBody
```

7

Ex. Compound If Statement

- Pay a certain tax rate depending on income level.

Income	Rate
0-47.500	22%
47.500 – 120.000	25%
120.000 -	35%

```
double rate;
if (income < 47500) rate = 0.22;
else if (income < 120000) rate = 0.25;
else rate = 0.35;
```

8

Enum Types

- Java supports an elegant approach to representing choices from a finite set by defining what is known as an enumerated type, or enum for short.
- These are types that are only allowed to take on values that come from a specified set of names. They are declared as follows:

  modifier enum name { *valueName0* , *valueName1* , . . . };
- Once defined, Day becomes an official type and we may declare variables or parameters with type Day. A variable of that type can be declared as:

```
public enum Day { MON, TUE, WED, THU, FRI, SAT, SUN };
public static Day today;
public static void main(String[] args) {
    today = Day.TUE;
  }
```

9

Switch Statements

- Java provides for multiple-value control flow using the switch statement.
- The switch statement evaluates an integer, string, or enum expression and causes control flow to jump to the code location labeled with the value of this expression.
- If there is no matching label, then control flow jumps to the location labeled "default."
- This is the only explicit jump performed by the switch statement, however, so flow of control "falls through" to the next case if the code for a case is not ended with a **break** statement

10

Switch Example

```
public enum Day { MON, TUE, WED, THU, FRI, SAT, SUN };
public static Day today;
public static void main(String[] args) {
    today = Day.TUE;
    switch (today) {
        case MON:
        System.out.println("This is tough.");
        break;
        case TUE:
        System.out.println("This is getting better.");
        break;
        case WED:
        System.out.println("Half way there.");
        break;
        case THU:
        System.out.println("I can see the light.");
        break;
        case FRI:
        System.out.println("Now we are talking.");
        break;
        default:
        System.out.println("Day off!");
    }
  }
```

11

Break and Continue

- Java supports a **break** statement that immediately terminate a while or for loop when executed within its body.
- Java also supports a **continue** statement that causes the current iteration of a loop body to stop, but with subsequent passes of the loop proceeding as expected.

12

Ex. Break and Continue

```
public class BreakAndContinue {

public static void main(String[] args)
{
int N = 15;
for (int i = 1; i <= N; i++)
{
    if (i>3 && i<12) continue;
    System.out.println(i);
}
}

}
```

```
1
2
3
12
13
14
15
```

```
public class BreakAndContinue {

public static void main(String[]
args) {
int N = 15;
for (int i = 1; i <= N; i++)
{
    if (i>3 && i<12) break;
    System.out.println(i);
}
}
}
```

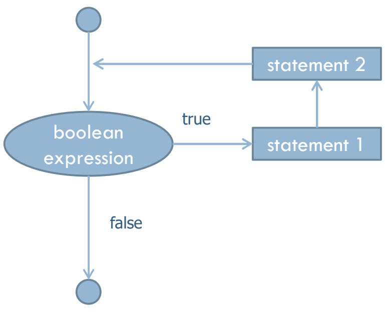
```
1
2
3
```

13

While Loops

- The while loop is a common repretition structure.
- Such a loop tests that a certain condition is satisfied and will perform the body of the loop each time this condition is evaluated to be true.

```
while (booleanExpression) {
        loopBody
}
```



14

Ex. While Statement

- Powers of 2

```
public class PowersOfTwo {
public static void main(String[] args) {
    // last power of two to print
    int N = 10;
    int i = 0; // loop control counter
    int v = 1; // current power of two
    while (i <= N) {
        System.out.println(i + " " + v);
        i = i + 1;
        v = 2 * v;
    }
}
}
```

```
0 1
1 2
2 4
3 8
4 16
5 32
6 64
7 128
8 256
9 512
10 1024
```

15

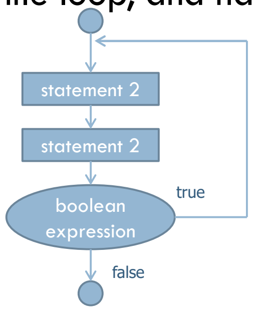
Do-While Loops

- Java has another form of the while loop that allows the boolean condition to be checked at the end of each pass of the loop rather than before each pass.
- This form is known as a do-while loop, and has syntax shown below:

do

loopBody

while (booleanExpression)



16

For Loops

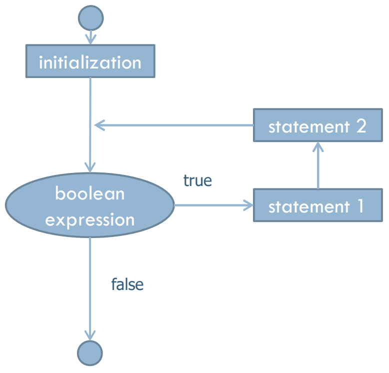
- The traditional **for**-loop syntax consists of four sections—an initialization, a boolean condition, an increment statement, and the body—although any of those can be empty.

for (initialization; booleanCondition; increment)

loopBody

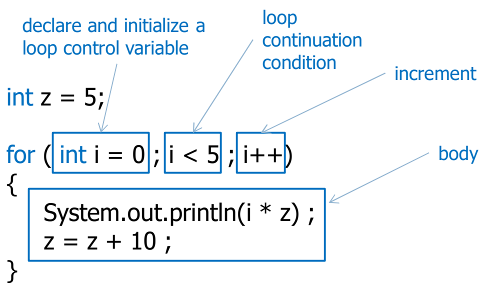
- Meaning:

```
{
    initialization;
    while (booleanCondition) {
        loopBody;
        increment;
    }
}
```



17

Ex. For Loops


declare and initialize a loop control variable
loop continuation condition
increment
body

```
int z = 5;
for ( int i = 0 ; i < 5 ; i++)
{
    System.out.println(i * z) ;
    z = z + 10 ;
}
```

18

Ex. For Loops

- Subdivisor of a ruler.

```
public class RulerN {
    public static void main(String[] args) {
        int N = Integer.parseInt(args[0]);
        String ruler = " ";
        for (int i = 1; i <= N; i++) {
            ruler = ruler + i + ruler;
        }
        System.out.println(ruler);
    }
}
```

Input	Output
1	" 1 "
2	" 1 2 1 "
3	" 1 2 1 3 1 2 1 "

19

Ex. For Loops

- Compute the sum of an array of doubles:

```
public static double sum(double[ ] data) {
double total = 0;
for (int j=0; j < data.length; j++) // note the use of length
    total += data[j];
return total;
}
```

- Compute the maximum in an array of doubles:

```
public static double max(double[ ] data) {
double currentMax = data[0]; // assume first is biggest (for now)
for (int j=1; j < data.length; j++) // consider all other entries
if (data[j] > currentMax) // if data[j] is biggest thus far...
currentMax = data[j]; // record it as the current max
return currentMax;
}
```

20

For-Each Loops

- Since looping through elements of a collection is such a common construct, Java provides a shorthand notation for such loops, called the **for-each** loop.
- The syntax for such a loop is as follows:

```
for (elementType name : container)
        loopBody
```

21

For-Each Loop Example

- Computing a sum of an array of doubles:

```
public static double sum(double[] data) {
   double total = 0;
   for (double val : data) // Java’s for-each loop style
      total += val;
   return total;
}
```

- When using a for-each loop, there is no explicit use of array indices.
- The loop variable represents one particular element of the array.

22

Ex. Loops

print largest power of two less than or equal to N	int v = 1;
	while (v <= N/2)
	   v = 2 * v;
	System.out.println(v);

compute a finite sum
(1 + 2 + ……. + N)
	int sum = 0;
	for (int i = 1 ; i <= N; i++)
	   sum += i;
	System.out.println(sum);

compute finite product
(1 X 2 X ……. X N)
	int product = 1;
	for (int i = 1 ; i <= N; i++)
	   product *= i;
	System.out.println(product);

print a table of function values	for (int i = 0 ; i <= N; i++)
	  System.out.println(i + “ “ + 2*Math.PI*i/N);

Simple Input

- There is also a special object, **System.in**, for performing input from the Java console window.
- A simple way of reading input with this object is to use it to create a **Scanner** object, using the expression

new Scanner(System.in)

```
import java.util.Scanner; // loads Scanner definition for our use
public class InputExample {
    public static void main(String[ ] args) {
        Scanner input = new Scanner(System.in);
        System.out.print("Enter your age in years: ");
        double age = input.nextDouble( );
        System.out.print("Enter your maximum heart rate: ");
        double rate = input.nextDouble( );
        double fb = (rate - age) * 0.65;
        System.out.println("Your ideal fat-burning heart rate is " + fb);
        input.close(); // close input stream
    }
}
```

java.util.Scanner Methods

- The Scanner class reads the input stream and divides it into tokens, which are strings of characters separated by delimiters.

hasNext( ):	Return **true** if there is another token in the input stream.
next( ):	Return the next token string in the input stream; generate an error if there are no more tokens left.
hasNext*Type*( ):	Return **true** if there is another token in the input stream and it can be interpreted as the corresponding base type, *Type*, where *Type* can be Boolean, Byte, Double, Float, Int, Long, or Short.
next*Type*( ):	Return the next token in the input stream, returned as the base type corresponding to *Type*; generate an error if there are no more tokens left or if the next token cannot be interpreted as a base type corresponding to *Type*.

25

H.W. 1.

- Write a short method in any language that counts the number of vowels in a given character string.
- Write a method that takes an array of float values and determines if all the numbers are different from each other (that is, they are distinct).
- Write a method that takes an array containing the set of all integers in the range 1 to 52 and shuffles it into random order. Your method should output each possible order with equal probability.

26