Chapter 7 - Iteration

Chapter Goals

- Program repetition statements – or loops – with the for, while, and do-while statements
- Learn potential pitfalls of infinite loops and off by one errors
- Understand nested loops
- Process input

Chapter 7

- Control statements
  - Already learned about selection statements
  - Now learn about repetition statements, or loop statements
- Repetition statements – repeat a block of code for a fixed number of times, or until some condition is met
- 3 types: while, do-while, and for

While statement

- While statements/loops, repeat a body of code until some condition is met
- This is helpful for certain problems such as:
  - Feed cat until it is full
  - Drink beer until pitcher is done
  - Get user input until they hit the Esc key
  - Play a game until someone wins

Example

- Add integers 1 through 100 (1+2+…+100)

```java
int sum = 0, number = 1; //Important to initialize
while ( number <= 100 ){
    sum = sum + number; //boolean expression
    number++; // what does this do?
}
```
if Flow Diagram

![if Flow Diagram](image)

while Flow Diagram

![while Flow Diagram](image)

Example

```java
int bottlesOfBeer = 99
while (bottlesOfBeer > 0){
    System.out.println(bottlesOfBeer + " on the wall");
    System.out.println(bottlesOfBeer + " on the wall");
    bottlesOfBeer--;
    System.out.println("Take one down, pass it around");
    System.out.println(bottlesOfBeer + " on the wall");
}
```

Compound Balance

- Problem: Want to calculate how many years my balance will take to appreciate to $20,000 given I start $10,000 and have a 5% interest rate

```java
int years = 0;
Scanner in = new Scanner(System.in);
System.out.print ("Enter target balance: ");
int targetBalance = in.nextInt();

while (balance < targetBalance)
{
    year++;
    double interest = balance * rate / 100;
    balance = balance + interest;
}
System.out.println("Your target will be achieved in "+ years + " years.");
```

![Flowchart of a while loop](image)
while (true){
  <statement>
}

- How long will this loop run?
- Why would we want to do this?
- Can we stop it?

Common Error 7.1
- Most common mistake – loop is never terminated
- <boolean expression> is always true
- Infinite loop – have to close program (Ctrl+c)

```java
int count = 1;
while (count != 10){
  count += 2;
}

int product = 0;
while (product < 500000){
  product *= 5;
}
```

Infinite loop
- Common cause – not advancing variable
  ```java
  int years = 0;
  while (years < 20){
    double interest = balance * rate / 100;
    balance = balance + interest;
  }
  ```
- Common cause – increment vs. decrement
  ```java
  int years = 20;
  while (years > 0){
    years++;
    double interest = balance * rate / 100;
    balance = balance + interest;
  }
  ```

Overflow
- Value of a variable exceeds precision
  ```java
  short s;
  while (s < 3000){
    s++;
  }
  ```
  ```java
  double count = 0;
  while (count != 1.0){
    count = count + 1.0/3.0;
  }
  ```
  ```java
  double count = 0;
  while (count != 1.0){
    count = count + 1.0/3.0;
  }
  ```
  ```java
  double count = 0;
  while (count != 1.0){
    count = count + 1.0/3.0;
  }
  ```

Underflow
- Real numbers are not always stored exactly, sometimes an approximation is needed
  ```java
  double count = 0;
  while (count != 1.0){
    count = count + 1.0/3.0;
  }
  ```

Off by one
- Another common error is to be off by one
  ```java
  int count = 1;
  while (count < 10){
    count++;
  }
  ```
  ```java
  int count = 0;
  while (count <= 10){
    count++;
  }
  ```

How many executions?
Off by one

- Be careful when counting
- Analogous to logic problems
- If I place a post every ten feet, how many posts do I need for a 100 ft fence?

Do-while statement

- The second repetition statement: **do-while**
  - while/loop/statement
  - while loops are use **pretest** format, where we test the boolean expression before executing anything
  - **do-while** is a **posttest** loop – we test the boolean after executing the loop

Syntax

```
do <single statement>
while ( <boolean expression> )
```

**OR**
```
do()
<statements>
while ( <boolean expression> )
```

Do-while vs while

- What does this posttest vs. pretest mean
  - A while loop **body** is not guaranteed to execute
    - while (false){…}
  - **do-while** body is guaranteed to execute **at least once**

**while** Flow Diagram

```
\[ condition \]
\[ true \]
\[ body \]
\[ false \]
```

**do-while** Flow Diagram

```
\[ body \]
\[ condition \]
\[ true \]
\[ false \]
```
Example

```java
int sum = 0, number = 1;
do{
    sum += number;
    number++;
} while (sum <= 1000000);
// Sums all numbers 1 through 1,000,000
```

Input

```java
double value;
do{
    System.out.println("Enter a positive number: ");
    value = in.nextInt();
} while (value <= 0);
```

While version

```java
count = 11;
do{
    System.out.println(count);
    count = count + 1;
} while (count < 5);
```

Avoid Repeat Code

```java
count = 0;
do{
    System.out.print("Enter score: ");
    score = in.nextInt();
    count++;
    if (count >= 20){
        System.out.println("Can't take more scores");
    } else if (score < 0){
        System.out.println("Invalid score");
    } else if (score == 0){
        System.out.println("User chooses to exit");
    }
} while (!(count >= 20 || score == 0 || score < 0 ));
```

```java
count = 0;
boolean repeat = true;
do{
    System.out.print("Enter score: ");
    score = in.nextInt();
    count++;
    if (count >= 20){
        System.out.println("Can't take any more scores");
        repeat = false;
    } else if (score < 0){
        System.out.println("Invalid score");
        repeat = false;
    } else if (score == 0){
        System.out.println("User chooses to exit");
        repeat = false;
    }
} while ( repeat )//Easier to understand
```
7.2 for loop

- Most common loop, mainly for **count-controlled** loops

```c
for(i = start; i <= end; i++)
{
    ...
}
```

### Syntax

```c
for(<initialization>; <boolean exp>; <update>)
    <single statement>
OR
for (<initialization>; <boolean exp>; <update>)
    <statements>
```

- Initialization occurs **only** the first time the loop is executed,
- Boolean expression is tested **before every** loop
- The increment operator is applied at the **end of each loop**

---

#### for Flow Diagram

![Flow Diagram for for loop](image)

#### Sum problem

- Saw it in while and do-while, here it is in for

```c
int i, sum = 0;
for (i = 1; i <= 100; i++)
{
    sum += i; // equivalent to sum = sum + i;
}
```

- **i** is a **control variable**, keeps track of number of repetitions

---

#### Interest Problem

```c
for(int i = 1; i <= n; i++)
{
    double interest = balance * rate/100;
    balance = balance + interest;
}
```
**Initialization**

```java
int sum = 0;
for (int i = 1; i <= 100; i++) {
    sum += i; // equivalent to sum = sum + 1;
}
```

- We can also declare `i` in the initialization, but `i` will be local to the for loop and not available outside.
  - Usually not an issue.
  - Can also leave initialization blank.

**Boolean Expression**

```java
int sum = 0;
for (int i = 1; i <= 100 && sum < 1111; i++) {
    sum += i; // equivalent to sum = sum + 1;
}
```

- Can test multiple conditions in boolean expression.
  - Is this still count controlled?

**Update**

```java
int sum = 0;
for (int i = 1; i <= 100 && sum < 1111; i += 2) {
    sum += i; // equivalent to sum = sum + 1;
}
```

- Can have any formula for incrementing.
  - Add only odd integers.
  - Decrease by 1, `i -= 1`.

**Sum problem**

```java
int i, sum = 0;
for (i = 1; i <= 100; i++) {
    sum += i; // equivalent to sum = sum + 1;
}
```

- `i` is set to 1 the first time the loop is executed.
- Before executing each time, check to see that `i <= 100` (like in while loop).
- Add 1 to `i` at the end of each cycle.
Legal for loops

- For loops can have many formats that are legal
- For (int i = 0; i <= 100; sum += i++);
- for(;;){
  ...
}
- System.out.println("Inputs: "); (x = in.nextDouble()) > 0; sum += x)
- count++;

Is this legal?

- for(int i = 0; i < 100; i++){i = 0; i < 100; i++)
  ...
}
- System.out.println(i);
- What if you want to know the value of i after loop is done

Mult. Table

- for (int i = 0; i <= 10; i++){
  for (int j = 0; j <= 10; j++){
    result = i * j;
    System.out.print(" "+ result);
  }
  System.out.println(" ");
}
- What will this output?
- What order will output in?
- How many times does each loop execute?

Scope

- Is this legal?
- for(int i = 0; i < 100; i++){
  ...
}
- System.out.println(i);
- What if you want to know the value of i after loop is done

7.4 Nested loops

- Recall from if-statements, any type of statement can be placed in the blocks or body
- In for loops, we can put an if statement, while loop, do-while loop, etc. inside the body
- Very common to have another for loop inside – a nested-for statement

Practice

- Write a loop to output the following pattern
  *
  **
  ***
  ****
  ...
  n rows
7.4 Sentinel Values

- Add integers 1 through 100 (1+2+...+100)
  ```java
  int sum = 0, number = 1; // Important to initialize
  while ( number <= 100 ){ // boolean expression
    sum = sum + number; // what does this do?
    number++; // what does this do?
  }
  ```
- **Count controlled** – the body is executed a fixed number of times

- **Sentinel-controlled loop** – executed repeatedly until a **sentinel** (designated value) is encountered

- **Sentinel value**: Can be used for indicating the end of a data set
  - 0 or -1 make poor sentinels; better use a meaningful value ('Q' for quit)

---

Loop and a half

```java
boolean done = false;
while(!done){
  System.out.print("Enter value, Q to quit: ");
  String input = in.next();
  if (input.equalsIgnoreCase("Q")){
    done = true;
  } else {
    double x = Double.parseDouble(input);
    ...
  }
}
```

How do we make this a loop?
Tips

- Symmetric vs. Asymmetric
  for(int i = 1; i <= n; i++)
  for(int i = 0; i < str.length(); i++)

- Counting iterations
  for(int i = a; i <= b; i++)

  How many executions?

Alternatives to loop and a half

- Can be confusing to read
  2 alternatives: test input in condition, or use break

  while(!(input = in.next()).equalsIgnoreCase("Q")){
    Process data
  }

break

boolean done = false;
while(!done){
  System.out.print("Enter value, Q to quit: ");
  String input = in.next();
  if (input.equalsIgnoreCase("Q")){
    break;
  } else {
    double x = Double.parseDouble(input);
  ...
  }
}

Code jumps

- break – exits loop
  Will immediately exit, just like for switch

- continue – will skip the rest of the statements in the loop and start next iteration of the loop

Spaghetti Code

- Many programmers avoid using these various jump statements
  break, continue, goto

- Can cause confusing code that often leads to harmful bugs

Which to choose?

- Count controlled
  for loops usually best

- Sentinel based loops
  while loops usually beset

- What about do-while?
  Priming reads, although can use a flag instead with while
7.5 Random Numbers

- In a simulation, you repeatedly generate random numbers and use them to simulate an activity.

```
Random number generator
Random generator = new Random();
int n = generator.nextInt(a); // 0 <= n < a
double x = generator.nextDouble(); // 0 <= x < 1
```

- Throw die (random number between 1 and 6)
```
int d = 1 + generator.nextInt(6);
```

Sequence

- If producing a random sequence, the sequence will be different every time.

```
Note: Not truly random (pseudorandom)
Formula used, but uses complicated factors to make it seem Random
```

Loop Invariant

- Loop invariant – a condition that is always true (beginning, after each iteration, and at the end).

```
Ex. Loop invariant: r*b^i = a^n
double r = 1, b = a;
int i = n;
while(i > 0){
    if(i%2 == 0){
        b = b*b;
        i = i--;
    } else {
        r = r*b;
        i--;
    }
}
```