Chapter 7

Iteration

Lesson page 7-1. Iteration

Activity 7-1-1 Introduction to the `while` loop

Question 1. The form of a while-loop is:

```
while ( boolean expression ) repetend
```

where the repetend is any statement.

Question 2. For a flow chart, look up "flaw chart" in ProgramLive's glossary.

Question 3. An iteration of a while-loop or for-loop is one execution of its repetend. The first iteration is iteration 0, the second is iteration 1, etc.

Question 4. The repetend of a while-loop or for-loop is the statement that is repeated over and over during execution. For example, the while-loop has the form

```
while ( boolean expression ) repetend
```

Question 5. The loop body of a while-loop or for-loop is the repetend. Repetend and loop body are synonyms.

Question 6. The loop condition of a while-loop is the boolean expression that is tested to see whether the loop should terminate:

```
while ( loop condition ) repetend
```

Question 7. Yes, the first iteration of a loop execution is called iteration 0.

Activity 7-1-2 The invariant of a `while` loop

Question 8. An invariant of a while-loop is a relation that is true before and after each iteration, i.e. whenever the loop condition is evaluated.

Question 9. The loop invariant is: `s` is the catenation of the first `k` capital letters.

Activity 7-1-3 Understanding the loop in terms of its invariant

Question 10. To falsify something is to make it false; to violate its truth. For example, you can falsify the relation `x = 5` by setting `x` to `6`.

Question 11. To truthify something is to make it true.

Question 12. A Hoare triple is a sequence of the form

> // { *precondition* }
> *statement*
> // { *postcondition* }

where the precondition and postcondition are relations. It has the meaning: if the precondition is true, then execution of the statement is guaranteed to terminate, and when it does, the postcondition will be true.

Question 13. The four points to check are:

1. Execution of the initialization must truthify the invariant.
2. Execution of the repetend must maintain the invariant.
3. Execution of the repetend must make progress toward termination.
4. The falsity of the loop condition together with the invariant must imply that the desired result is true.

Activity 7-1-4 Summing the first `n-1` positive numbers

Question 14. A fresh variable is a new variable, one that has not yet been used in the context under consideration.

Question 15. A relation is an expression that is true or false; to complement a relation is to change its value –from true to false or false to true. In Java, the complement operator is `!`.

Question 16. The five steps in developing a loop are:

1. Find the loop invariant.
2. Develop initialization to truthify the invariant.
3. Find the loop condition, by finding a relation that describes when the loop can terminate and complementing that relation.
4. Begin writing the repetend to make progress toward termination.
5. Change the repetend to maintain the loop invariant.

Question 17. `3+...+4` equals `7`.

Question 18. `3+...+3` equals `3`.

Question 19. `3+...+2` equals `0`.

Question 20. To develop the invariant, we made a copy of the postcondition and replaced a variable (`n`) by a fresh variable (`k`).

Activity 7-1-5 Executing the summing loop

Question 21. For the relation `x = 1+...+(k-1)` to be true when `k = 3`, `x` should be `3`.

Activity 7-1-6 Understanding a loop in terms of its invariant

Question 22. `n-1` iterations are performed. For example, when `n=2`, `1` iteration is performed.

Question 23. When `k = n-1`, one more iteration will be performed.

Question 24. When `k = n`, no more iterations will be performed.

Question 25. The number of iterations still to be performed is `n-k`.

Activity 7-1-7 Summing the first `n` positive numbers

Question 26. Both loops terminate with `k = n`.

Question 27. The loops calculate different sums because they begin with `k` having different values —one `0` and the other `1`. Also, the assignments to `x` within the repetends are different.

Question 28. The initializations and the repetends cause the loops to do different things.

Question 29. In the first loop, the range of `k` has size `n`; in the second loop, `n+1`.

Question 30. The ranges of `k` contain one more number than is summed because `k's` last value is used to flag termination of the loop. When `k` reaches that last value, the loop body is not executed.

Activity 7-1-8 Exercises on loops

Lesson page 7-2. Several examples of loops

Activity 7-2-1 Developing invariants

Question 1. The underlined words are: postcondition; fresh variable.

Question 2. The technique works well when a range of the integers is to be processed.

Question 3. Here is the annotated loop:

```
initialization
// {precondition: Q}
// {invariant: P}
while (C) {
    // {C and P}
    repetend
    // {P}
}
// {!C and P}
// {postcondition: R}
```

Question 4. Relation `R1` is weaker than relation `R2` if `R1` is **true** in the same states (and perhaps more states) as `R2`.

Question 5. Relation `R1` is strong than relation `R2` if `R2` is weaker than `R1`.

Question 6. Conjunction is operator `&&` in Java; `b && c` is **true** iff both `b` and `c` are **true**.

Question 7. A conjunct is an operand of a conjunction.

Activity 7-2-2 The roach explosion

Question 8. `!(a <= b)` is `a > b`.

Question 9. No, they are not equal, because in `5*x/4`, operator `/` denotes integer division and not mathematical division.

Activity 7-2-3 Exponentiation

Question 10. False: $b^{2\times i} = (b^2)^i$

Question 11. True.

Question 12. `x` is `b` and `y` is `c`.

Question 13. Here's the answer:

```
// return b^c, given c >= 0
public static int exp(int b, int c) {
    int x= b;
    int y= c;
    int z= 1;
    // {invariant: z*x^y = b^c and 0 <= y <= c}
    while (y != 0) {
        if (y % 2 == 0)
            { x= x*x; y= y/2; }
        else
            { z= z*x; y= y-1; }
    }
    // {postcondition: z = b^c}
    return z;
}
```

Activity 7-2-4 The spiral

Question 14. No answer can be given.

Lesson page 7-3. Loop schemata

Activity 7-3-1 A counting-loop schema

Question 1. A schema is a diagrammatic depiction of a typical or average situation; a generalized presentation, framework of reference, outline, or plan.

Question 2. A loop schema is a generalized loop (possibly with initialization) for performing some abstract task, like reading in and processing some numbers —here, "processing" is the abstract task.

Question 3. A natural number is one of the integers 0, 1, 2, ...

Activity 7-3-2 Counting the w's

Question 4. Variable `n` was replaced by expression `s.length()`.

Question 5. Processing `k` means to add `1` to `x` if `s[k]` is `'c'`.

Question 6. The invariant is the postcondition but with `n` replaced by `k`.

Activity 7-3-3 Testing primality

Question 7. An integer is prime if it is greater than `1` and is divisible by no positive integer other than `1` and itself.

Question 8. To handle the range `2..p-1`, we changed `0` to `2` and `n` to `p` in the postcondition, invariant, initialization, and loop condition.

Question 9. Use `j % i == 0` to check whether `i` divides `j`.

Activity 7-3-4 Loop schema for reading/processing nonzero integers

Question 10. If the first input value is `0`, `0` iterations are performed.

Activity 7-3-5 Summing the nonzero input

Question 11. Here is the loop:

```
// read nonzero integers followed by 0
// and print every integer that is divisible by 5
    int v= JLiveRead.readInt();
    // inv: v contains the last value read and
    //     input values before v that were divisible
    //     by 5 have been printed
    while (v != 0) {
        if (v % 5 == 0)
            { System.out.println(v); }
        v= JLiveRead.readInt();
    }
```

Question 12. Here is the loop:

```
// read nonzero integers followed by 0 and print
// every integer that is divisible by its predecessor
    int v= JLiveRead.readInt();
    int pred= 0;
    // inv: v contains the last value read, and
    //      pred contains the integer that precedes
    //          v (0 if none), and
    //      input values before v that were divisible by
    //      their predecessor have been printed
    while (v != 0) {
        if (pred != 0 && v % pred == 0)
            { System.out.println(v); }
        pred= v;
        v= JLiveRead.readInt();
    }
```

Lesson page 7-4. The for loop

Activity 7-4-1 The `for` loop

Question 1. Underlined phrases: initialized; incremented; decremented; end.

Question 2. When the loop has a loop counter.

Activity 7-4-2 Syntax and semantics of the `for` loop

Question 3. The syntax is:

```
for ( <initialization> ; <condition> ; <progress> )
    <repetend>
```

Question 4. The translation is:

```
b= p>1;
int k= 2;
while (k != p) {
    b= b && (p % k == 0);
    k= k+1;
    k++;
}
```

Activity 7-4-3 Developing a `for` loop

Question 5. Here is the filled-in code:

```
// Print 9, 8, down to 2
    // {inv: 9, 8, down to k+1 have been printed}
```

```
        for (int k= 9; k != 1; k= k-1) {
            System.out.println(k);
        }
// {R: 9, 8, down to 2 have been printed}
```

Question 6. Here is the filled-in schema

```
// Process natural numbers 0..(n-1), where n >= 0
// {invariant: 0..k-1 have been processed}
for (int k= 0; k != n; k= k+1) {
    Process k;
}
// {postcondition: 0..(n-1) have been processed}
```

Activity 7-4-4 Exercises on `for` loops

Lesson page 7-5. Making progress and stopping

Activity 7-5-1 The bound function

Question 1. The underlined terms are: upper; iterations left; decreases; `F > 0`; loop condition; `F > 0`.

Activity 7-5-2 The stopping condition `k!=n`

Question 2. Suppose for some reason, probably because of an error in the repetend, that `k` becomes greater than `n`. With the loop condition `k < n`, the error may be undetected until shipped to the customer. With the loop condition, `k != n` the error will be detected the first time it is tested because the loop won't terminate.

Activity 7-5-3 Using the condition `k<n`

Question 3. The operator `&&` is called conjunction.

Question 4. The other conjunct is `k != n+1`.

Activity 7-5-4 Off-by-one errors

Question 5. Any true-false statement is a relation. For example, "The moon is pink" and "`x` is at least `5`".

Question 6. An off-by-one error is an error that causes a loop to iterate one more time or one less time than it should.

Question 7. The underlined phrases are: loop condition; postcondition; complement.

Question 8. Suitable loop conditions:

```
!(k == s.length()-1)
k != s.length()-1
```

Question 9. Suitable loop conditions:
`!(a == 0)`
`a != 0`

Question 10. Suitable loop conditions:
`!(b == 50)`
`b != 50`

Lesson page 7-6. Miscellaneous points about loops

Question 1. In Java, the empty statement consists simply of a semicolon `;` . Its execution does nothing (but extremely fast).

Question 2. The loop will perform infinitely many iterations: the repetend is the empty statement so `i` is never incremented.

Activity 7-6-1 Counting primes

Question 3. A program-development strategy in which, at each step, some abstract statement (written in English, say), is implemented —its implementation could consist of a sequence of statements, some of which could themselves be written in English.

Question 4. Here is the translated code:

```
// Set x to the # of primes in 2..99
    int x= 0;
    // {inv: x = # primes in 2..i-1}
    for (int i= 2; i != 100; i= i+1){
        if (isPrime(i)) { x= x+1; }
    }

// = "n is a prime"
public static boolean isPrime(int n) {
    if (n < 2)
        return false;
    // inv: no number in 2..k-1 divides n
    for (int k= 2; k != n; k= k+1) {
        if (n % k == 0) return false;
    }
    return true;
}
```

Activity 7-6-2 About nested loops

Question 5. Statement-comment: a comment in a program that is a statement written in English (or other language), with its implementation indented underneath it.

Question 6. 10

Activity 7-6-3 How not to design loops

Question 7. The student's strategy was: write the program first and fill in statement-comments and loop invariants and other comments afterward.

Question 8. Here are some (not all) of the strategies:

1. Use a disciplined programmng style.
2. Reduce debugging time and effort by being so careful that you don't introduce bugs into the program in the first place.
3. Use good and standard conventions for naming entities (e.g. variables, methods).
4. Use good conventions for indentation.
5. Write a clear and precise specification of a method, mentioning all the parameters, before you write the method body. If you want to change the method, change its specification before changing its body.
6. Write clear and precise specifications of variables before writing code that uses them.
7. Write a loop invariant before writing the loop.
8. Develop a loop using the checklist for understanding a loop.
9. Use schemas to reduce your programming time.
10. Write a simple but correct program first; then see how to make it more efficient.