**Checkpoint Answers Chapter 5**

**5.1**

The condition count < 100 is always true at point A.

The condition count < 100 can be both true and false at point B.

The condition count < 100 is always false at point C.

**5.2**

The problem with initializing guess to zero is that the number that shall be guessed will sometimes be zero. This will cause the while to not even run one iteration and the player will never get to guess the number.

**5.3**

**(a)**  
The loop body will be run an infinite number of times. There will be no output because i is 1 all the time.

**(b)**  
The loop body will be run an infinite number of times. There will be no output because i is 1 all the time.

**(c)**  
The loop body will be run nine times and the output will be:

3

5

7

9

**5.4**

The variable max will hold the largest number entered and number will hold the last number entered so the output will be:

max is 5

number 0

**5.5**

This program will start with x equal to 80000000 and x is then increased by one again and again until it overflows after reaching 2147483647, x will be -2147483648 after the overflow. The while loop will now terminate and the output will be:

x is -2147483648

**5.6**

The program in this checkpoint will take numbers as input and keep track of the biggest number in the variable called "max". The last entered number is stored in the variable called "number".

The output when running the program with the input

2 3 4 5 0

will be:

max is 5

number 0

**5.7**

A do-while loop will always execute the body of the loop at least one time, this is not true for a while loop.

The code from the checkpoint rewritten using a do-while loop:

Scanner input = new Scanner(System.in);

int sum = 0;

int number;

do {

System.out.println("Enter an integer (the input ends if it is 0)");

number = input.nextInt();

sum += number;

} while (number != 0);

##5.8 ##

for (int i = 0; i < 10; ++i) {

sum += i;

}

for (int i = 0; i < 10; i++) {

sum += i;

}

The above two loops will result in the same value in sum. This is so because the control variable i will have identical value, when used in the loop-continuiation-condition, regardless of ++i or i++ is used to increment i.

##5.9## The three parts usually present in a for loop control section are

1. Initial action used to initialize control variables.
2. Loop continuation condition that controls if the body of the loop will run or not, will usually be based on the control variables.
3. After iteration action which usually will alter the value of the control variable.

A loop that prints the numbers for 1 to 100:

for (i = 1; i <= 100; i++) {

System.out.println(i);

}

##5.10##

import java.util.Scanner;

public class CheckPoint\_05\_10 {

public static void main(String[] args) {

Scanner input = new Scanner(System.in);

int number, sum = 0, count;

for (count = 0; count < 5; count++) {

number = input.nextInt();

sum += number;

}

System.out.println("sum is " + sum);

System.out.println("count is " + count);

}

}

The program in this checkpoint will sum integers until the control variable in the loop reaches the value five 5. This means that if the input to the program is

2 3 4 5 0

so will the output be

sum is 14

count is 5

##5.11## The loop in this check point will iterate indefinitely and repeat the execution of the instructions in the body on each iteration.

##5.12## A variable that is declared in a for loop control section can not be used after the loops exists.

##5.13## A for loop, a while loop, and a do-while loop, that all do the same thing.

long sum = 0;

for (int i = 0; i <= 1000; i++)

sum = sum + i;

long sum = 0;

int i = 0;

while (i <= 1000) {

sum += i;

i++;

}

long sum = 0;

int i = 0;

do {

sum += i

i++;

} while (i <= 1000);

##5.14## **(a)**  
n iterations  
**(b)**  
n + 1 iterations  
**(c)**  
n - 5 iterations  
**(d)**  
(n-5)/3 iterations, rounded upwards if n-5 is not divisible by 3.

n equal to 12 will for example result in

(12 - 5) / 3 = 7 / 3 = 2.33

which shall be rounded upwards resulting in 3 loop iterations.

##5.15## A for loop can be converted into a while loop.

Some benefits of for loops are:

* For certain type of problem, a for loop can be easier to construct and easier to read than the corresponding while loop.
* It is a very good (perhaps the best) choice for counting loops.
* All the control variables are nicely collected into one place when using for loops.

##5.16## A while loop can always be converted into a for loop.

Below follows an illustration where a given while loop have been converted into a corresponding for loop.

int i = 1;

int sum = 0;

while (sum < 10000) {

sum = sum + i;

i++;

}

sum = 0;

for (int i=1; sum < 1000; i++) {

sum = sum + i;

}

##5.17## Fixed version of the code in this checkpoint follows. Had to guess a little on some of the fixes since there was no specification and instead just a piece of semi-random nonsense code.

public class Test {

public static void main(String[] args) { // missing static

int sum = 0; // sum was not defined in original code

for (int i = 0; i < 10; i++) // removed a semicolon here

sum += i;

int j = 10; // j was not defined

int i = 5; // i was not defined

if (i < j) // removed a semicolon

System.out.println(i); // added missing semicolon

else

System.out.println(j);

while (j < 10) // removed semicolon

{

j++;

}

do {

j++;

} while (j < 10); // added semicolon

}

}

##5.18## **(a)**  
The problem with this code is a missing semicolon after the do-while loop at row 8.  
**(b)**  
The problem with this code is that the semicolon at line 3 shall be removed.

##5.19## When i is 0, the println statement is executed 0 times.  
When i is 1, the println statement is executed 1 time.  
When i is 2, the println statement is executed 2 times.  
.  
.  
.  
When i is 9, the println statement is executed 9 times.

So, the total is

0 + 1 + 2 + 3 + 4 + 5 + 6 + 7 + 8 + 9 = 45

##5.20## **(a)**  
Output:

0 0 1 0 1 2 0 1 2 3

**(b)**  
Output:

\*\*\*\*

\*\*\*\*

2 \*\*\*\*

3 2 \*\*\*\*

4 3 2 \*\*\*\*

**(c)**  
Output:

1xxx2xxx4xxx8xxx16xxx

1xxx2xxx4xxx8xxx

1xxx2xxx4xxx

1xxx2xxx

1xxx

**(d)**  
Output:

1G

1G3G

1G3G5G

1G3G5G7G

1G3G5G7G9G

##5.21## The suggested optimization is incorrect since it will not spot the gcd when entering two identical numbers.

##5.22## The code will not compile if we remove the cast. This is because the code is semi dangerous since information might be lost when we convert an int to a char. An int can hold a wider range of numbers than a char.

But no information will be lost in this case because the numbers are always small enough to fit in a char.

So there is no problem with our code but we still need to make a cast to a char to indicate to compiler that we are aware of what we are doing.

##5.23## The loop will in each iteration divide the variable "decimal" by 16. The result will be truncated since this is how integer division works in Java. The execution stops when "decimal" is 0.

Assume that "decimal" starts at 245:

245 / 16 = 15

15 / 16 = 0

There will hence be 2 iterations in this case.

Assume that "decimal" starts at 3245:

3245 / 16 = 202

202 / 16 = 12

12 / 16 = 0

There will hence be 3 iterations in this case.

##5.24## The keyword break can be used in a loop to exit the loop.

The keyword continue is used in a loop to terminate the current iteration and move execution directly to the loop control section.

The keyword break is used in the following code:

int balance = 10;

while (true) {

if (balance < 9)

break;

balance = balance - 9;

}

System.out.println("Balance is " + balance);

Balance will become 1 after the first iteration. This causes a break out of the loop and the output will be:

Balance is 1

The keyword continue is used in the following code:

int balance = 10;

while (true) {

if (balance < 9)

continue;

balance = balance - 9;

}

System.out.println("Balance is " + balance);

The loop in this second example will never end because continue is mistakenly used instead of break.

##5.25## The problem with the while loop version (see the book) is that the varible called "i" will not be incremented as it should when it is divisible by three. This is because the continune will cause an jump to the beginning of the loop.

We need to add the increment of "i" in one extra place:

int i = 0;

while (i < 4) {

if (i % 3 == 0) {

i++;

continue;

}

sum += i;

i++;

}

##5.26## The program TestBreak in Listing 5.12 rewritten so that the break is no longer used:

public class TestBreak {

public static void main(String[] args) {

int sum = 0;

int number = 0;

while (number < 20 && sum < 100) {

number++;

sum += number;

}

System.out.println("The number is " + number);

System.out.println("The sum is " + sum);

}

}

The program TestContinue in Listing 5.13 rewritten so that the continue is no longer used:

public class TestContinue {

public static void main(String[] args) {

int sum = 0;

int number = 0;

while (number < 20) {

number++;

if (number != 10 && number != 11)

sum += number;

}

System.out.println("The sum is " + sum);

}

}

##5.27## **(a)**  
The second print statement will be executed after the break. The output will be:

1

2

1

2

2

3

**(b)**  
The increment of j will be executed after the continue. The output will be:

1

2

1

2

2

3

* © 2017 GitHub, Inc.
* [Terms](https://github.com/site/terms)
* [Privacy](https://github.com/site/privacy)
* [Security](https://github.com/security)
* [Status](https://status.github.com/)
* [Help](https://help.github.com/)
* [Contact GitHub](https://github.com/contact)
* [API](https://developer.github.com/)
* [Training](https://training.github.com/)
* [Shop](https://shop.github.com/)
* [Blog](https://github.com/blog)