

CHAPTER 6

Iteration

TOPIC OUTLINE

- 6.1 `while` Loops
- 6.2 `for` Loops
- 6.3 Nested Loops
- 6.4 Off-by-1 Errors
- 6.5 Random Numbers and Simulations
- 6.6 Loop Invariants (AB only)

■ Topic Summary

6.1 `while` Loops

Many applications require us to execute statements (or blocks of statements) multiple times. For example, averaging grades for all of the students in a particular course, printing a calendar of twelve months, processing a payroll for many employees, and any number of other applications involve doing the same task more than once. One way to implement this type of algorithm is to use a `while` loop. The general form of a `while` loop is

```
while (condition)
{
    statement;
}
```

Example 6.1 uses a `while` loop to sum the positive integers less than 10.

Example 6.1

```
int sum = 0;
int x = 1; // Initializes loop control variable x.
while (x < 10) // Checks end condition.
{
    sum += x;
```

```

        x++;    // Loop control variable updated.
    }
    // At the termination of the loop, x has value 10.

```

The three important pieces of a loop are initializing the loop control variable, testing for the end condition, and advancing the value of the loop control variable so that its value approaches the end condition. If you don't change the variable value within the loop, you create an endless loop.

Example 6.2

```

int sum = 0;
int i = 1;
while (i < 10)    // Infinite loop
{
    sum += i;    // WRONG! i is not incremented.
}

```

Example 6.2 will add `i` to `sum` repeatedly but, because the value of `i` starts at 1 and never changes, `i` is always less than 10. The end condition of the loop is never satisfied. This is called an *infinite loop*. *Common Error 6.1* discusses infinite loop errors.

Example 6.3 is designed to accept non-negative integers (points scored in basketball games) and return the sum of those points. The user must enter a negative value to end input.

Example 6.3

```

public static int getSum()
{
    int total = 0;
    int points;
    System.out.println("Enter game points, enter a negative number
        to quit.");
    points = number input by user;
    while (points >= 0)
    {
        total += points;
        System.out.println("Enter game points, enter a negative
            number to quit.");
        points = number input by user;
    }
    return total;
}

```

An alternate method for accomplishing this same task using a type of infinite `while` loop implementation is called a *loop and a half* and is discussed in *Advanced Topic 6.4* in your text.

6.2 for Loops

Another method for iterative control is a `for` loop. If you know exactly how many times a block is to be executed, a `for` loop is often the best choice. The general form of a `for` loop is

```

for (initialization; test; update)
{
    statement;
}

```

Examples 6.4 and 6.5 add the integers 1 to 10 inclusive. The order in which the integers are added is different but the result is the same.

Example 6.4

```

sum = 0;
for (int i = 1; i <= 10; i++)
{
    sum += i;
}
// At the termination of this loop, i has value 11.

```

Example 6.5

```

sum = 0;
for (int i = 10; i >= 1; i--)
{
    sum += i;
}
// At the termination of this loop, i has value 0.

```

The loop control variable, *i*, is initialized. The loop continues while the test is true. The variable *i* is updated (incremented or decremented) by 1 each time the loop executes. Example 6.6 gives the equivalent `while` loop for Example 6.5

Example 6.6

```

sum = 0;
int i = 10;
while (i >= 1)
{
    sum += i;
    i = i - 1;
}

```

The initialization, the test, and the update in the `for` header should be related (see Example 6.7). The code to update the loop control variable should occur in the `for` header only, not in the body of the loop (see Example 6.8).

Example 6.7

```

int j = 1;
int k = 8;
for (int i = 0; j <= 10; k = k + 2)    // BAD STYLE!
                                        // Variables are not related.
{
    j += 2;
    i = i + 1;
}

```

```

        System.out.println(i + " " + j + " " + k);
    }

```

Example 6.8

```

for (int i = 0; i <= 10; i++)
{
    sum += i;
    i = i + 1;    // WRONG! Update should occur only in for header.
}

```

It is common to see the loop control variable declared in the header as it is initialized.

```

for (int i = 1; i <= 10; i++)
{
    Statement;
}

```

A variable declared in this manner cannot be accessed outside the loop. We say that the *scope* of the variable extends to the end of the loop. This is explained in *Advanced Topic 6.2* in your text.

A `for` loop should be used when a statement, or group of statements, is to be executed a *known* number of times. *Quality Tip 6.1* discusses this. Use a `for` loop for its intended purpose only.

6.3 Nested Loops

Loops can be nested (one loop inside another loop). One typical application of a nested loop is printing a table with rows and columns.

Example 6.9

```

final int MAXROW = 5;
final int MAXCOL = 4;
for (int row = 1; row <= MAXROW; row++)
{
    for (int col = 1; col <= MAXCOL; col++)
    {
        System.out.print("*");
    }
    System.out.println();
}

```

This program prints:

```

****
****
****
****
****

```

6.4 Off-by-1 Errors

It is common to have off-by-1 errors when writing `while` loops and `for` loops. Consider the following problem.

Problem

Count the number of times the letter “a” occurs in the word “mathematics”.

Solution Outline

- Loop through each letter of “mathematics”.
- Each time an “a” is encountered, increment a counter.

Examples 6.10 and 6.11 illustrate two correct solutions.

Example 6.10

```
String word = "mathematics";
int count = 0;
String lookFor = "a";

for (int i = 0; i < word.length(); i++)
{
    if (word.substring(i, i + 1).equals(lookFor))
    {
        count++;
    }
}
```

Example 6.11

```
String word = "mathematics";
int count = 0;
String lookFor = "a";

int j = 0;    // Init to 0
while (j < word.length())    // Note: < not <=
{
    String temp = word.substring(j, j + 1);
    if (lookFor.indexOf(temp) != -1)
    {
        count++;
    }
    j++;    // Increment outside if
}
```

Possible Off-by-1 Errors

- The loop variable should start at 0, not 1, because the index of the first letter in a `String` is 0.
- The loop test should be `< word.length()` not `<= word.length()` because the last letter in a `String` has index `length() - 1`.
- The variable used as the counter should be initialized to 0, not 1.

Off-by-1 errors are discussed in *HOWTO 6.1* in your text.

6.5 Random Numbers and Simulations

Random number generators are useful in programs that simulate events. The Java library's `Random` class implements a *random number generator*. This random number generator can produce random integers and random floating-point (double) numbers.

The AP testable subset includes the `java.util.Random` methods listed in Table 6.1.

Table 6.1

class java.util.Random

<i>Method</i>	<i>Method Summary</i>
<code>int nextInt(n)</code>	Returns a random integer in the range from 0 to $n - 1$ inclusive.
<code>double nextDouble()</code>	Returns a random floating-point number between 0 (inclusive) and 1 (exclusive).

Example 6.12 demonstrates basic calls to the methods of the `Random` class.

Example 6.12

```
Random generator = new Random();
    // Constructs the number generator.

int randInt = generator.nextInt(10);
    // Generates a random integer from 0-9 inclusive.

double randDouble = generator.nextDouble();
    // Generates a random double from 0(inclusive) to 1(exclusive).
```

When writing programs that include random numbers, keep in mind that two `Random` objects created within the same millisecond will have the same sequence of random numbers. For example, if the following declarations are executed in the same millisecond,

```
Random generator1 = new Random();
Random generator2 = new Random();
```

`generator1` and `generator2` will generate the same sequence of random numbers.

It is a better idea to share a single random number generator in the entire program. We rarely want to generate identical sequences of random numbers!

Example 6.13 uses the `Random` class to simulate rolling two 6-sided dice until doubles are rolled (the same number appears on both dice). The variable `count` counts the number of rolls it takes to roll doubles.

Example 6.13

```
Random die = new Random();
int count = 1;
while (die.nextInt(6) != die.nextInt(6))
{
```

```

        count++;
    }
    // count is the number of rolls it takes to roll doubles.
    System.out.println("Doubles were rolled on roll #" + count);

```

6.6 Loop Invariants (AB only)

A loop invariant is a statement that is true before a loop executes, at the start of each iteration of the loop, and after the loop terminates. Loop invariants can be used to help explain algorithms that are not completely obvious. They are also statements that can be used for proving the correctness of loops. Correctness proofs, though not part of the AP CS subset, are discussed in *Random Fact 6.2* of your text.

Example 6.14

This segment of code is intended to count the number of letters in `word` that are vowels (a, e, i, o, or u).

```

String vowel = "aeiou";
String word = some String;
int count = 0;
// loop invariant: count = number of vowels in word.substring(0, i)
for (int i = 0; i < word.length(); i++)
{
    if (vowel.indexOf(word.substring(i, i + 1)) >= 0)
        count++;
}

```

The `for` loop visits each letter of `word`. If the letter is a vowel, `indexOf` returns a value between 0 and 4 and `count` is incremented. After each execution of the loop, `count` is the number of vowels in the substring looked at so far. The loop invariant is true before the loop executes, after each iteration of the loop, and after the loop terminates. After the loop terminates, `count` is the number of vowels (a, e, i, o, u) in `word`.

A more detailed example of a loop invariant is discussed in *Advanced Topic 6.8* in your text.

■ Topics That Are Useful But Not Tested

- It is often useful to read test data from a *file* instead of typing it for each execution of a program. You can then create a test data file once and reuse it. Reading data from a file is discussed in *Productivity Hint 6.1* in your text.
- Sometimes an input line may contain several items of input data. The `StringTokenizer` class can be used to break the input line into separate strings. This is discussed in Section 6.4.2 of your text.
- It is sometimes useful to traverse the characters in a string using the `charAt` method of the `String` class. This method uses the primitive type `char` which is not tested on the AP CS Exams.

- Sometimes you may want to execute the body of a loop at least once and perform the loop test at the end of the loop. A `do` loop serves this purpose and is explained in *Advanced Topic 6.1* in your text.

■ Things to Remember When Taking the AP Exam

- When implementing a `while` loop be sure to initialize the loop control variable before the loop begins and to advance the variable's value inside the loop.
- Be careful not to put a semicolon after a `for` loop header or a `while` statement.

```
for (i = 1; i < 10; i++);    // WRONG!

while (i < 10);           // WRONG!
```

- Be sure to use braces where needed for the body of a loop. If in doubt, use the braces.
- Check the initialization and the test condition in your loops. It is common to have an off-by-1 error when writing loops.
- Be careful when implementing tables with nested loops. The outer loop controls the rows of the table and the inner loop controls the columns.
- **(AB only)** Know the definition of a loop invariant. You may be asked to choose the correct loop invariant on a multiple-choice question.

■ Key Words

You should understand the terms below. The AP CS Exam questions may include references to these terms. The citations in parentheses next to each term identify the page numbers where it is defined and/or discussed in *Computing Concepts with Java Essentials*, 3rd ed., and *Big Java*.

file (253)	nested loops (244)	simulation (262)
<code>for</code> loops (236)	random number	variable scope (241)
infinite loops (231)	generator (262)	<code>while</code> (228)
loop invariants (267)	random numbers (262)	
(AB only)		

■ Connecting the Detailed Topic Outline to the Text

The citations in parentheses identify where information in the outline can be located in *Computing Concepts with Java Essentials*, 3rd ed., and *Big Java*.

- `while` Loops (228–232)
- `for` Loops (236–244)
- Nested Loops (244–245)
- Off-by-1 Errors (232–233)

- Random Numbers and Simulations (262–267)
- Loop Invariants (**AB only**) (267–268)

■ Practice Questions

Multiple Choice

1. Consider the following segment of code.

```
String word = "mathematics";
String vowels = "aeiou";
String tempWord = "";
String newWord = "";

for (int i = 0; i < word.length(); i++)
{
    tempWord = word.substring(i, i + 1);
    if (vowels.indexOf(tempWord) >= 0)
    {
        newWord += tempWord;
    }
}
```

After the loop is terminated, what is the value of `newWord`?

- a. “mathematics”
 - b. “mthmtcs”
 - c. “aeai”
 - d. “aei”
 - e. the empty string
2. Consider the `Die` class as defined below.

```
public class Die
{
    // Constructs an s-sided die.
    public Die(int s) {...}

    // Simulates a throw of the die, returning a random integer from
    // 1 to s (the number of sides) inclusive.
    public int cast() {...}

    // Private stuff goes here
}
```

The following declaration is made.

```
Die d1 = new Die(6);
```

82 Computing Concepts with Java Essentials Advanced Placement CS Study Guide

Which segment of code returns the number of rolls it takes to roll double 1s (both dice have the value 1)?

I.

```
count = 1;
while (!(d1.cast() == 1 && d1.cast() == 1))
{
    count++;
}
return count; // Double 1s rolled on roll #count.
```

II.

```
count = 1;
while (true)
{
    if (d1.cast() == 1 && d1.cast() == 1)
    {
        return count;
    }
    count++; // Double 1s rolled on roll #count.
}
```

III.

```
count = 1;
while (d1.cast() != 1 || d1.cast() != 1)
{
    count++;
}
return count; // Double 1s rolled on roll #count.
```

- a. I
- b. II
- c. III
- d. II and III only
- e. I, II, and III

3. Consider the following code segment.

```
for (int i = 0; i < 5; i++)
{
    for (int j = 0; j < 5; j++)
        System.out.print(i * j % 5);
    System.out.println();
}
```

What is the output produced?

- a.

```
01234
12340
23401
34012
40123
```

b. *12345*
12345
12345
12345
12345

c. *00000*
01234
23401
34012
40123

d. *00000*
01234
02413
03142
04321

e. *00000*
00000
00011
00112
00123

4. The following triangle design is to be printed.

4
33
222
1111

Which of the following code segments correctly prints this design?

I.

```
for (int i = 4; i >= 1; i--)
{
    for (int j = 4 - i + 1; j >= 1; j--)
        System.out.print(i);
    System.out.println();
}
```

II.

```
for (int i = 1; i <= 4; i++)
{
    for (int j = 1; j <= i; j++)
        System.out.print(4 - i + 1);
    System.out.println();
}
```

III.

```
for (int i = 1; i <= 4; i++)
{
    for (int j = 4; j >= 4 - i + 1; j--)
        System.out.print(4 - i + 1);
    System.out.println();
}
```

- I only
- II only
- III only
- I and II only
- I, II, and III

5. Consider the following code segment.

```
int n = some integer value;
int a = 0 ;
while (n > 0)
{
    a += n % 10;
    n /= 10;
}
System.out.println("answer is :" + a);
```

Which of the following statements best describes the result of executing this code?

- 0 is printed.
- The number of digits in *n* is printed.
- The sum of the digits in *n* is printed.
- The original value of *n* is printed.
- An endless loop results.

6. Consider the code segment below.

```
String word = "computer";
String tempWord = "";
for (int i = word.length() - 1; i >= 0; i--)
{
    tempWord = word.substring(i, i + 1) + tempWord;
}
System.out.println(tempWord);
```

What will be printed when the code is executed?

- computer*
- retupmoc*
- erteutpumpomco*
- Nothing will be printed. *tempWord* is an empty *String*.
- An error message.

7. Consider the following code segment in which `IO.readInt()` is a call to a method that reads an integer.

```
String tempSequence = "";
int x = IO.readInt();
int y = IO.readInt();
while (y >= x)
{
    tempSequence += x;
    x = y;
    y = IO.readInt();
}
tempSequence += x;
System.out.println(tempSequence);
```

What is the output if the series of integers being input is: 1 1 2 3 5 4 7 8?

- a. 1 1 2 3
- b. 1 2 3 5
- c. 1 1 2 3 5
- d. 1 2 3 5 4 7 8
- e. 1 1 2 3 5 4 7 8

8. Consider the following code segment where `IO.readWord()` is a call to a method that reads a `String`.

```
String word = IO.readWord();
for (int i = 0; i < word.length() - 1; i++)
{
    if (word.substring(i, i + 1).equals(word.substring(
        i + 1, i + 2)))
    {
        return true;
    }
}
return false;
```

Which of the following describes its results?

- a. Returns true if any letter in `word` is repeated, false otherwise.
- b. Returns true if the first and second letters are the same, false otherwise.
- c. Returns true if any two consecutive letters are the same, false otherwise.
- d. Always returns true.
- e. Always returns false.

9. Given the following declarations:

```
String vowel = "aeiou";
String word = some String value;
int count = 0;
```

Which of the following segments of code accurately counts the number of letters in `word` that are vowels (a, e, i, o, or u)?

- I.

```
for (int j = 0; j < vowel.length(); j++)
{
    String temp = vowel.substring(j, j + 1);
    if (word.indexOf(temp) != -1)
    {
        count++;
    }
}
```
- II.

```
for (int j = 0; j < word.length(); j++)
{
    String temp = word.substring(j, j + 1);
    if (vowel.indexOf(temp) != -1)
    {
        count++;
    }
}
```
- III.

```
for (int i = 0; i < word.length(); i++)
{
    for (int k = 0; k < vowel.length(); k++)
    {
        if (word.substring(i, i + 1).equals(vowel.substring(
            k, k + 1)))
        {
            count++;
        }
    }
}
```
- a. I only
b. II only
c. III only
d. I and III only
e. II and III only

10. Consider the following code segment.

```
int p = 1;
int i = 1;
int n = some positive integer value;
// loop invariant
while (i <= n)
{
    p = p * i;
    i++;
}
```

Which of the following statements is a correct loop invariant?

- a. $i < n$
- b. $0 < i < n$
- c. $p = i!$
- d. $p = (i - 1)!$
- e. $p = n^i$

11. Consider the following code segment.

```
for (int i = 0; i < 5; i++)
    for (int j = i; j < 5; j++)
        System.out.print("*");
```

How many stars will be printed?

- a. 5
- b. 10
- c. 15
- d. 20
- e. 25

12. Consider the following code segment.

```
int count = 0;
for (int x = 0; x < 3; x++)
    for (int y = x; y < 3; y++)
        for (int z = y; z < 3; z++)
            count++;
```

What is the value of `count` after the code segment is executed?

- a. 81
- b. 27
- c. 10
- d. 9
- e. 6

13. Each segment of code below correctly calculates the sum of the integers from 1 to n inclusive and stores this value in the integer variable `sum`.

I.

```
int sum = 0;
for (int i = 1; i <= n; i++)
    sum += i;
```

II.

```
int sum = 0;
int i = n;
while (i > 0)
{
    sum += i;
    i--;
}
```

III.

```
int sum = (n + 1) * n / 2;
```

88 Computing Concepts with Java Essentials Advanced Placement CS Study Guide

For large values of n , which of the following statements is true about the execution efficiency of the above code segments?

- I is more efficient than II and III because it sums the numbers in increasing numeric order.
- II is more efficient than I and III because it sums the numbers beginning with the largest value.
- III is more efficient than I and II because there are fewer operations.
- I and II are more efficient than III because they are easier to understand.
- I, II, and III always operate with the same execution efficiency.

Questions 14 and 15 refer to the following code segment.

```
int n = some integer value;
int x = 0;

while (n > 0)
{
    n = n / 2;
    x++;
}
```

14. If n refers to the original integer value, which of the following statements is always true after the loop is terminated?

- $2^{x-1} = n$
- $2^x \geq n$
- $2^x \leq n$
- $2 \cdot x = n$
- $2 \cdot n = x$

15. If n has the value of 32 before the loop is executed, how many times is the statement

```
x++;
```

executed?

- 2
- 5
- 6
- 8
- 16

Free Response Questions

1. Consider the `Investment` class, whose incomplete definition is shown below.

```
public class Investment
{
    // Constructs an Investment object from a starting balance and
```

```

// interest rate.
public Investment(double aBalance, double aRate)
{
    balance = aBalance;
    rate = aRate;
    years = 0;
}

// Interest is calculated and added to balance at the end of
// each year for y years.
public void waitForYears(int y)
{
    // Code goes here
}

// Returns the current investment balance.
public double getBalance() {...}

// Returns the number of years this investment has accumulated
// interest.
public int getYears() {...}

// Calculates the amount of interest earned in y years
// compounded n times per year at an annual interest rate, rate.
// balance is updated.
public void compoundTheInterest(int y, int n)

// Other methods here

private double balance;
private double rate;
private int years;
}

```

- a. Write the method `waitForYears` for the `Investment` class that will calculate the amount of interest earned in a given number of years. Interest is accumulated and the balance is updated at the end of each year. The table below contains sample data values and appropriate updates.

<i>Starting Balance</i>	<i>Interest Rate</i>	<i>Years (y)</i>	<i>Ending Balance</i>
100.00	5%	10	162.88
500.00	4%	20	1095.56
1000.00	5%	10	1628.89

Implement the method `waitForYears` using the following header.

```
public void waitForYears(int y)
```

- b. Write the method `compoundTheInterest` for the `Investment` class that will calculate the amount of interest earned in y years compounded n times per year at an annual interest rate of $r\%$.

The method `compoundTheInterest` will update the balance appropriately. For example, if your original balance is \$100.00 and you invest your money in a bank with an annual interest rate of 5% compounded quarterly (4 times a year) and you leave your money in the bank for 1 year, the interest and the ongoing balance is calculated as described below.

If your original balance is \$100.00, then

After the first quarter your balance is $100 + (100)(.05/4) = \$101.25$.

After the second quarter your balance is $101.25 + (101.25)(.05/4) = \106.52 .

After the third quarter your balance is $106.52 + (106.52)(.05/4) = \107.85 .

After the fourth quarter, your balance is $107.85 + (107.85)(.05/4) = \109.20 .

Your balance at the end of the year is \$109.20.

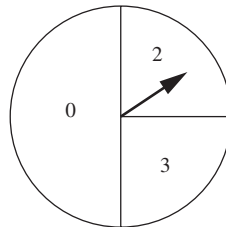
The private instance variable, `balance`, should be updated appropriately. The table below contains sample data values and appropriate updates.

<i>Starting Balance</i>	<i>Interest Rate</i>	<i>Years (y)</i>	<i>Times Per Year (n)</i>	<i>Ending Balance</i>
100.00	5%	10	12	164.70
500.00	4%	20	4	1108.35
1000.00	5%	10	4	1643.62

Use the method header below to write `compoundTheInterest`.

```
// Balance is compounded n times annually for y years.
public void compoundTheInterest(int y, int n)
```

2. A modified game of roulette is played as follows. A player has a purse that contains coins. The player bets one coin (chosen randomly from the purse) on a number on the roulette wheel. This modified roulette wheel is shown below.



The spinner spins. If the spinner lands on “0”, the player gets nothing back. If the spinner lands on “2”, the player gets the wagered coin back and an additional coin of equal value. If the spinner lands on “3”, the player gets 3 coins of the same value as the coin that was bet.

In this particular version of roulette, you either win big or you lose big because the spinning continues until you double the money in your purse or you have no coins left in your purse! The classes used are:

- The `Spinner` class, used to “spin” the roulette wheel
- The `Coin` class, which defines a coin
- The `Purse` class that holds the coins
- The `Game` class that simulates the roulette game

Incomplete definitions of these classes are below.

```
// The Spinner class is used to simulate the spin.
public class Spinner
{
    // Constructs a spinner with s choices.
    public Spinner(int s) {...}

    // Returns an integer from 0 to the number of choices - 1
    // inclusive.
    public int spin() {...}

    // Private stuff here
}
```

```
public class Coin
{
    // Constructs a coin.
    // aValue the monetary value of the coin
    // aName the name of the coin
    public Coin(double aValue, String aName) {...}

    // Returns the coin value.
    public double getValue() {...}

    // Returns the coin name.
    public String getName() {...}

    // Other methods and private stuff here
}
```

```
public class Purse
{
    // Constructs an empty purse.
    public Purse() {...}

    // Adds a coin to the purse.
    public void add(Coin aCoin) {...}

    // Removes a coin from the purse.
    public Coin removeCoin() {...}

    // Returns the total value of the coins in the purse.
    public double getTotal() {...}

    // Returns the number of coins in the purse.
    public int coinCount() {...}

    // Other methods and private stuff goes here
}
```

```
public class Game
{
    // Game constructor
    // Creates the roulette wheel by constructing the
    // appropriate Spinner.
    public Game(Purse myPurse)
    {
        // Code goes here
    }

    // Simulates a spin returning a 0, 2, or 3 as defined by the
    // roulette wheel pictured in the problem.
    public int spinTheWheel()
    {
        // Code goes here
    }

    // Simulates the game of roulette as follows:

    // Until the purse total is twice its original value
    // or there are no coins left in the purse,
    // removes a coin from the purse, and
    // updates the number of coins in purse according to the
    // winnings.

    // Returns the total value in the purse.
    public double playRoulette(Purse myPurse)
    {
        // Code goes here
    }

    private Spinner myWheel;
}
```

- a. Write the `Game` constructor. This constructor should construct a `Spinner` that will be able to appropriately simulate the wheel shown in the problem description above. Notice that all outcomes on the wheel do not have equal probabilities.
- b. Write the method `spinTheWheel` that will call `spin` to generate a random number and then return the appropriate number on the roulette wheel. Notice that all outcomes on the wheel do not have equal probabilities. Use the header below when writing `spinTheWheel`.

```
public int spinTheWheel()
```

- c. Write the method `playRoulette` that will bet (remove a coin from the purse), spin the roulette wheel, and accumulate appropriate winnings by adding coins to the purse until the player either wins big (doubles the original purse value) or loses big (has no coins left in the purse). Write `playRoulette` using the method header below.

```
public double playRoulette(Purse myPurse)
```