

# WES-CS GROUP MEETING #7

## Exercise 1: If and Switch Statements

Use the yellow and pink cards for this exercise. Each yellow card has a code fragment that includes an *if* statement, and each pink card has a code fragment that includes a *switch* statement. Your job is to match each *if*-fragment with an equivalent *switch*-fragment (not all of the *switch*-fragments have a matching *if*-fragment). For the *switch*-fragments that are similar but not equivalent to the *if*-fragments, explain what is wrong.

## Exercise 2: Loops

### Part (a)

Write code that prints each minute between 1:00pm and 2:59pm using one or more loops; i.e., your code should print:

```
1:00pm
1:01pm
...
1:59pm
2:00pm
2:01pm
2:59pm
```

Swap your code with your neighbor and see if either of you can find any errors. Once you think you both have correct code, try typing it in, compile it, and see if it works.

### Part (b)

Write code to print all of the 15-minute interval times between 11:00am and 1:15pm (i.e., 11:00am, 11:15am, ..., 1:00pm, 1:15pm). Note that noon is 12:00pm.

## Exercise 3: More Loops

Remember the *Artist* class we used for our first set of exercises; it provided the following methods:

`drawLineDown( int length )`

Draw a vertical line of the given length (in inches), starting from the current position and going straight down. The current position is changed to be at the bottom end of the line.

`drawLineUp( int length )`

Draw a vertical line of the given length, starting from the current position and going straight up. The current position is changed to be at the top end of the line.

`drawLineRight( int length )`

Draw a horizontal line of the given length, starting from the current position and going straight to the right. The current position is changed to be at the right end of the line.

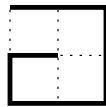
`drawLineLeft( int length )`

Draw a horizontal line of the given length, starting from the current position and going straight to the left. The current position is changed to be at the left end of the line.

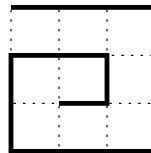
The pictures below show four *clockwise square spirals*, each of which starts at the upper-left corner of the square, and stops when drawing the next line would run into a line that's already been drawn (the dotted lines are just there to show you a grid of inches).



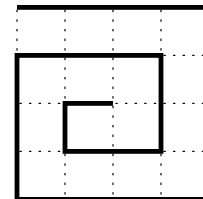
1-by-1 inch square



2-by-2 inch square



3-by-3 inch square



4-by-4 inch square

Write the code that would make an *Artist* object called *picasso* draw each of these spirals. Look for the pattern, then write a method with an *int* parameter *n* that uses a loop to make *picasso* draw a spiral in an *n*-by-*n* inch square. Test your code using the *Artist* class.

## Exercise 4: The AlarmClock class

For this exercise we're going to design a new class that uses the *Time* class you worked on last time. If you didn't get to work on the *Time* class, don't worry, you'll be able to do this exercise anyway.

The incomplete *Time* class (a little different from the one we used last week) is given below. Notice that the *Time* class stores its time in 24-hour format: it has two fields called *hours* and *minutes*; *hours* is an integer between 0 and 23 that represents the hours part of the time, and *minutes* is an integer between 0 and 59 that represents the minutes part of the time.

```
public class Time {
    private int hours;    // 0 <= hours <= 23
    private int minutes; // 0 <= minutes <= 59

    // constructor
    public Time(int hr, int min) {
        hours = hr;
        minutes = min;
    }

    // print24HourTime
    public void print24HourTime() {
        System.out.print(hours + ":");
        if (minutes < 10) System.out.print("0");
        System.out.println(minutes);
    }

    // timeRemaining
    public int timeRemaining( Time otherTime ) {
        // part (a)
    }
}
```

### Part (a)

Write the *timeRemaining* method, which returns the number of minutes that would have to be added to the time represented by the *Time* object whose method is called, to get to the time represented by parameter *otherTime*. For example, if you have a *Time* object *now* that represents 11:30, and another *Time* object *appointment* that represents 13:45, then the method call *now.timeRemaining( appointment )* should return 135 because it takes 135 minutes (which is two hours and fifteen minutes) to get from 11:30 to 13:45. The method call *appointment.timeRemaining( now )* should return 1305, because it takes 1305 minutes (which is 21 hours and 45 minutes) to get from 13:45 to 11:30 (the next day).

## Part (b)

Now we're ready to define the *AlarmClock* class. This class is for your team leader who sometimes parties late into the night and oversleeps the next morning.

The *AlarmClock* class should have one field (a *Time* object that is the wake-up time), a constructor, whose parameter is the wake-up time, plus two public methods:

1. A *doAlarm* method, that keeps checking the current time until it is equal to the wake-up time (see below for how to get the current time). Then it beeps, and finishes.
2. A *main* method that gets the wake-up time from the user of the program (the hours, in 24-hour format, and the minutes, with a space between them; for example: 13 25), creates a *Time* object to represent the wake-up time, uses the *AlarmClock* constructor to create a new *AlarmClock* object, tells the user what the current time is and how many minutes remain until the alarm will ring, and finally calls the *AlarmClock* object's *doAlarm* method.

To find out what the current time is you can use this code (you will need to include *java.util.\**):

```
Calendar cal = new GregorianCalendar();
int hour = cal.get(Calendar.HOUR_OF_DAY);
int min = cal.get(Calendar.MINUTE);
```

This will set variables *hour* and *min* to the current, 24-hour time (for example, if the time is 1:26pm, *hour* will be 13 and *min* will be 26).

To make the computer beep, use `System.out.print('\u0007');`

Write the *AlarmClock* class and test it. If you want to improve your *AlarmClock* class, try the following:

1. Your team leader is kind of a heavy sleeper, and so is not likely to be woken up by a single-beep alarm clock. Modify the *doAlarm* method so that it beeps once per second for 60 seconds.
2. Make it easier for your team leader to set the alarm by allowing the wake-up time to be specified in 12-hour format. For example: 1:23pm instead of 13 23.

```
if (x == 0) y = 1;
else if (x == 1) y = 2;
else y = 3;
```

```
if (x == 0) y = 1;
else if (x == 1) y = 2;
```

```
if ( x == 4 ) {
    x += 4;
} else if ( (x > 4) && (x <= 7) ) {
    x += 7;
}
```

```
if ( x == 1 || x == 3 ) {
    System.out.println("Good Morning!");
} else {
    System.out.println("Good Afternoon!");
}
```

```
if (x == 0 && y == 0) doSomething();
else if (x == 0 && y == 1) doAnotherThing();
else doSomethingElse();
```

```
if ( character != ' ' ) action1();  
else action4();
```

```
if ( x == 8 ) x = 0;  
else if ( x*2 == 8 ) x = 2;  
else if ( x - 1 == 13 ) x = 3;
```

```
if ( x > 2 && x < 4 ) x = 10;  
if ( x <= 0 || x > 4 ) x = 100;
```

```
if ( i > 0 && i < 4 ) x = 10;  
else if ( i == 4 ) x = 20;  
else if ( i <= 0 || i > 7 ) x = 40;  
else if ( i > 4 && i <= 7 ) x = 30;
```

```
switch (x) {  
    case 0: y = 1;  
           break;  
    case 1: y = 2;  
           break;  
    default: y = 3;  
}
```

```
switch (x) {  
    case 0: y = 1;  
           break;  
    case 1: y = 2;  
           break;  
}
```

```
y = 3;
```

```
switch (x) {  
    case 0: y = 1;  
           break;  
    case 1: y = 2;  
           break;  
}
```

```
switch (x) {  
    case 0: y = 1;  
           break;  
    case 1: y = 2;  
           break;  
    default: y = 0;  
}
```

```
switch ( x ) {  
    case 4 : x = x + 4; break;  
    case 5 :  
    case 6 :  
    case 7 : x = x + 7; break;  
}
```

```
switch(x) {  
    case 5:  
    case 6:  
    case 7: x = x + 7;  
           break;  
    default: x = x + 4;  
}
```

```
switch(x) {
    case 1:
    case 3: System.out.println("Good Morning!");
           break;
    default: System.out.println("Good Afternoon!");
}
```

```
switch(x) {
    case 1:
    case 3: System.out.println("Good Morning!");
}
System.out.println("Good Afternoon!");
```

```
switch (x) {
    case 0: switch (y) {
                case 0: doSomething();
                       break;
                case 1: doAnotherThing();
            }
            break;
    default: doSomethingElse();
}
```

```
switch (x) {  
    case 0: if (y==0) doSomething();  
           else doAnotherThing();  
           break;  
    default: doSomethingElse();  
}
```

```
switch( character ) {  
    case ' ': action4(); break;  
    default:  action1();  
}
```

```
switch( character ) {  
    case ' ': action1(); break;  
    default:  action4();  
}
```

```
switch ( x ) {  
    case 8: x = 0; break;  
    case 4: x = 2; break;  
    case 14: x = 3; break;  
}
```

```
switch ( x ) {  
    case 8: x = 0;  
    case 4: x = 2;  
    case 14: x = 3;  
}
```

```
switch ( x ) {  
    case 1:  
    case 2:  
    case 4: break;  
    case 3: x = 10; break;  
    default : x = 100;  
}
```

```
switch ( x ) {  
    case 3: x = 10; break;  
    default : x = 100;  
}
```

```
switch ( i ) {  
    case 1 :  
    case 2 :  
    case 3 : x = 10; break;  
    case 4 : x = 20; break;  
    case 5 :  
    case 6 :  
    case 7 : x = 30; break;  
    default : x = 40;  
}
```

```
switch ( i ) {  
    case 1 :  
    case 2 :  
    case 3 : x = 10;  
  
    case 4 : x = 20;  
  
    case 5 :  
    case 6 :  
    case 7 : x = 30;  
  
    default : x = 40;  
}
```