

WES-CS GROUP MEETING #5

Exercise 1: Boolean conditions

Play a game using the decks of yellow and green cards. Each green card has an English sentence that is either true or false. Each yellow card has one of three symbols: !, &&, or || (meaning *not*, *and*, or). The game is played as follows:

- Each player starts with 6 cards, 3 yellow and 3 green.
- When it's your turn, if you can make a logical formula that evaluates to true using at least 3 of your cards, put down the formula and draw new cards of the same colors as the ones you used. Otherwise, trade in two of your cards for two new cards of the same color.
- The first person to put down at least 12 cards wins.

The precedences of the logical operators are as follows:

!	highest precedence
&&	next highest precedence
	lowest precedence

Exercise 2: The Code game

An incomplete *CodeGame* class is on the next two pages.

First, get some practice with code-tracing: Trace the code (starting with the *main* method), using a real die and a real coin for the *roll* and *toss* methods.

Now work together to write the actual code for the *roll* and *toss* methods. There are several ways to generate pseudo-random numbers. You may have seen this done using the *random* method of the *Math* class:

`Math.random()` returns a double value greater than or equal to 0.0 and less than 1.0. Returned values are chosen pseudorandomly with (approximately) uniform distribution from that range.

You can also use the *nextInt* method of the *Random* class (which is in *java.util*):

```
Random ran = new Random();
```

`ran.nextInt(n)` returns an int value greater than or equal to zero and less than n (with uniform distribution in that range).

```

class CodeGame {
    // data members
    private int points; // the number of points
    private static final boolean HEADS = true; // for coin toss
    private static final boolean TAILS = false; // for coin toss

    // constructor
    public CodeGame() {
        points = 0;
    }

    // getPoints: return the current number of points
    public int getPoints() {
        return points;
    }

    // roll: return a number 1 - 6 (representing a roll of a die)
    private int roll() {
        // not yet implemented
    }

    // toss: return HEADS or TAILS representing a coin toss
    private boolean toss() {
        // not yet implemented
    }

    // main method
    public static void main( String[] args ) {
        CodeGame game = new CodeGame();
        game.play();
        System.out.println( "Your score is " + game.getPoints());
    }
}

```

```

// play: play the game
public void play() {
    int oneRoll; // the current roll of the die; 1 - 6
    boolean oneToss; // the current toss of the coin; HEADS or TAILS

    oneRoll = roll();
    while (oneRoll <= 3) {
        points++;
        oneRoll = roll();
    }

    if ( oneRoll == 6 || oneRoll == 4) {
        points++;
    } else {
        points--;
    }

    oneRoll = roll();
    while (oneRoll > 0) {
        points += 2;
        oneRoll--;
    }

    oneToss = toss();
    if ( oneToss==HEADS ) {
        points *= 2;
    } else {
        points += 2;
    }

    while ( oneToss != HEADS ) {
        oneRoll = roll();
        points -= oneRoll;
        oneToss = toss();
    }

    oneRoll = roll();
    switch ( oneRoll ) {
    case 1:  points += points / 2;
            break;
    case 2:  points += roll();
            break;
    case 3:  oneToss = toss();
            if ( oneToss == HEADS ) points += 10;
            break;
    default: points = points + roll() * oneRoll - oneRoll * 2;
            break;
    }
}
}

```

Exercise 3: Algorithms

Part (a).

For this exercise, we'll say that a nickname is formed by taking any non-empty prefix of a name (the first one or more letters) followed by "ie". For example, this rule gets us the following:

Name	Nickname	Not a Nickname
Susan	Susie	Suzy
Martin	Martie	Smartie
Ann	Annie	nnie

It also gets us some nonsense nicknames:

Name	Nickname
Susan	Sie
Martin	Martinie
Ann	Anie

but we won't worry about that.

Your job is to write an algorithm (in English or in Java) to determine whether one string (called *nick*) is a nickname for another string (called *name*). For example, if *nick* is "Susie" and *name* is "Susan" then your algorithm should say that *nick* **is** a nickname for *name* (or your Java code should return *true*). If *nick* is "Susie" and *name* is "George" then your algorithm should say that *nick* is **not** a nickname for *name*.

Whether you write your algorithm in English or in Java, you should make use of the *String* methods to find out how long *nick* and *name* are, and to extract and compare substrings. Think about which pieces of the two strings you need to look at, and what the values of those pieces need to be.

Part (b).

Now we'll write an algorithm that figures out whether the string in *word1* is a **subsequence** of the string in *word2*. String *word1* is a subsequence of *word2* if all of the letters in *word1* occur in *word2* in the same order, but perhaps with some other letters in between. For example

<i>word1</i>	<i>word2</i>	<i>word1</i> is a subsequence of <i>word2</i> ?
win	wisconsin	yes
wise	wisconsin	no
snow	wisconsin	no
sin	wisconsin	yes

Again, you can write your algorithm in English or in Java, but in either case you should use the String methods.

Exercise 4: Logical thinking

Arrange 9 chocolates in 3 rows with 4 in the first row, 3 in the second row, and 2 in the third row.

This is a game for two players. You win by picking up the last chocolate. When it's your turn, you play by taking as many chocolates as you like from any row (you may take the whole row if you like) but from one row only. This is the game of NIM and is actually a logical puzzle, because the first player can always win. Figuring out how to do that is pretty tough, but you should be able to figure out some good strategies.

Divide into groups of 2 and try playing the game a few times, alternating who goes first. Think about what leads to a win. Can you find some rules for good configurations to present to your opponent? It might help to start at the end of the game and work backwards; i.e., under what conditions are you guaranteed to win when it's your turn? What conditions can you present to your opponent that forces him/her to leave you a guaranteed win?

If you get tired of playing the game this way, try playing that the person who takes the last chocolate *loses*.

The Earth is flat

Madison is the capital of Wisconsin

In Java, * has higher precedence than +

In Java, + has higher precedence than -

The moon is made of green cheese

17 is a prime number

There is not a DDR machine in Memorial Union

The dorms get Cartoon Network

Red and blue make purple

A chestnut tree makes acorns

A giraffe has a black tongue

Ducks can't fly

We are learning C++ in CS302

A soccer ball is round

The sun moves around the Earth

The UW-Madison was founded in 1492

There are 10 WES-CS groups

This room is CS 1240

Miss Muffet is afraid of spiders

The time now is 3:33pm

The sun sets in the west

Java programs start executing in a class named Main

The following code will compile: `boolean b = "true";`

An if statement will repeat until the condition is false

&&

&&

&&

&&

&&

&&

&&

&&

&&

&&

&&

&&

||

||

||

||

||

||

||

||

||

||

||

||

!

!

!

!

!

!

!

!

!

!

!

!