AP Computer Science Name: ______________________________
Karel Chapter 5 Notes (3)

MORE COMPLEX TESTS

It may not be a trivial matter to have a robot make two or more tests at the same time. Programming languages provide the capability to make multiple tests within an IF or an IF/ELSE instruction. We can do this but we must be clever with our programming.

Example 1: Multiple Requirements

Suppose you want to execute a block of code only if…

- karel is facing west
- karel's right side is blocked
- karel's left side is blocked
- karel's front is clear
- there is at least one beeper on the corner

Following these requirements we must plan an instruction that will test all of these conditions simultaneously. If we do what seems logical we might try to write something like this:

```
if (      facingWest()
     AND rightIsBlocked()
     AND leftIsBlocked()
     AND frontIsClear()
     AND nextToABeeper() )
{ …}
```

This seems very logical, but there is one major problem. If we use a sequence of nested IF instructions to do the job the result will be very ugly.

```
if ( facingWest() )
{
     if( ! rightIsClear() )
     {
          if( ! leftIsClear() )
          {
               if ( frontIsClear() )
               {
                    if( nextToABeeper() )
                    {
                         <instruction>
                    }
               }
          }
     }
}
```

If we trace this, we will find that all of the tests must evaluate to true before karel can execute <instruction>.

&& Java symbol for "AND"

|| Java symbol for "OR"

Fortunately, Java and the robot programming language have an operator for AND, but it is spelled && with two ampersand characters. So we can actually say:

```
if (      facingWest()
     && rightIsBlocked()
     && leftIsBlocekd()
     && frontIsClear()
     && nextToABeeper()
     )
{    …    }
```

Another way to build complex tests is to define new predicates. Suppose we would like to write

```
if (nextToABeeper() && leftIsBlocked() )     {     …     }
```

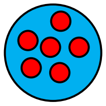
This can be done if we write a new predicates in the class in which we need such a test. There is no real need for this, given the && operator, but it is instructive to examine it. For example:

```
public boolean nextToABeeper_AND_leftIsBlocked()
{
     if(nextToABeeper())
     {
          if( ! leftIsClear())
          {
               return true;
          }
          else
          {
               return false;
          }
     }
     return false;
}
```

This can be simplified to:

```
public void nextToABeeper_AND_leftIsBlocked()
{
     if(nextToABeeper())
     {
          return ! leftIsClear();
     }
     return false;
}
```

Example 2:   Truth Tables (Logic)

	Red	Blue	Red **AND** Blue &&	Red **OR** Blue ||
				
🔴				
🔵				
🟢				

Short-Circuit Evaluation:

Note: The && operator has a higher precedence than the || operator. This means that in a sequence of terms separated by these operators the && operators are applied first. If you wish it otherwise, you can use parenthesis, just as in arithmetic expressions using addition and multiplication. You also work left to right among operators of the same precedence.

Example 3:

Write a new predicate that tests if a robot is `nextToABeeper()` and `leftIsBlocked()`.

Review:

- The IF instruction allows a robot to decide whether to execute or skip entirely the block of instructions within the THEN clause.

- The IF/ELSE instruction allows a robot to decide whether to execute the block of instructions in the THEN clause or the ELSE clause.

- Nesting these instructions allows karel to make more complex choices if required.