Making Decisions

- How do we solve tasks in which every particular of a task is not specifically known?
  - A robot needs the ability to survey its immediate environment and make decisions about what to do next
  - The `if` and the `if/else` provide robots with decision making abilities

- Robot programs contain several different kinds of instructions
  - Messages to robots, either primitives or new instructions
  - Constructors (specifications on how to build a robot)
  - Control statements: `if` and `if/else` are the first examples of these
**General form**

```plaintext
if ( <test> )
{
    <instruction-list>
}
```

- Robot determines whether `<test>` is true or false
- `<instruction-list>` is a sequence of any valid robot commands that will be performed when the robot determines the `<test>` is true
- Nothing will happen when the `<test>` is false
Example of if statement

```
// in a method
if (this.nextToABeeper()) {
    this.pickBeeper();
}
this.turnLeft();

// in a main task block
if (Karel.nextToABeeper()) {
    Karel.pickBeeper();
}
Karel.turnLeft();
```

- A beeper will be picked up only if there is a beeper on the same corner as the robot
  - Dependent upon the results of the test

- The robot will turn left, regardless of whether a beeper is on the corner or not
  - Independent of the results of the test
Conditions that a VPI Robot can test

class VPIRobot extends ur_Robot ()
{
    boolean frontIsClear();
    boolean nextToABeeper();
    boolean nextToARobot();
    boolean facingNorth();
    boolean facingSouth();
    boolean facingEast();
    boolean facingWest();
    boolean anyBeepersInBeeperBag();
    boolean running();
}

Suppose we want a negative form of a predicate?

We can precede a predicate with the negative operator!

```java
if (!Karel.next-To-A-Beeper())
{
    // stuff
}
```

<table>
<thead>
<tr>
<th>A</th>
<th>!A</th>
</tr>
</thead>
<tbody>
<tr>
<td>true</td>
<td>false</td>
</tr>
<tr>
<td>false</td>
<td>true</td>
</tr>
</tbody>
</table>
The if/else instruction

- **General form**

\[
\text{if ( <test> ) } \{ // note lack of semicolon} \\
\quad <\text{instruction-list-1}> \\
\}
\]

\[
\text{else } \{ // note lack of semicolon} \\
\quad <\text{instruction-list-2}> \\
\}
\]

- **Robot determines whether** <test> **is true or false**
- **<instruction-list-1>** is a sequence of any valid robot commands that will execute when the robot determines the <test> is true
- **<instruction-list-2>** will execute when the <test> is false
New predicate instructions

- We could create a new subclass of the robot class and provide a new predicate method

```java
class myRobot extends VPIRobot
{
    // assume constructor

    boolean frontIsBlocked()
    {
        return ! frontIsClear();
    }

    // more stuff
}
```
In the block of the definition of a new predicate we need to indicate what value is to be returned. We need to use the reserved word `return`, followed by an expression. The returned reference value replaces the message in the invoking statement. In a predicate method the value of the expression must be `true` or `false`. Java supports a simple data type specialized for representing logical values. `boolean` type variables can have either of two values: `true` or `false`. The identifiers `true` and `false` are Java reserved words. In Java, in order to ask a question, a program makes an `assertion` which is evaluated to either true or false at run-time.
void task()
{

    myRobot R = new myRobot(1, 1, North, 0);

    if (R.frontIsBlocked())
    {
        // stuff
    }

    // and so on
}
Suppose we needed to determine if there is a wall to the right or left?

```java
boolean leftIsClear() {
    turnLeft();
    if (frontIsClear()) {
        turnRight(); // assume definition
        return true; // method terminates here
    }
    turnRight();
    return false;
}
```

Return statement immediately terminates the method
Nested if Instructions

- Written with an if instruction nested inside the THEN or else clause of another if

- As an example:

```java
if ( <test> ) {
    if ( <test> ) {
        <instruction-list-1>
    } else {
        <instruction-list-2>
    }
} else {
    <instruction-list-2>
}
```
When to Use an if Instruction

- 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 or skip the block of instructions within the THEN clause or the else clause.

- Nesting these instructions allows robots to make more complex choices.