Java Programming for Beginners

CEMC Summer Institute for High School CS Teachers
Byron Weber Becker
bwbecker@cs.uwaterloo.ca

Session Goals

(group discussion)

What's an Object?

(group discussion; role play with a robot object)

Programming a Robot

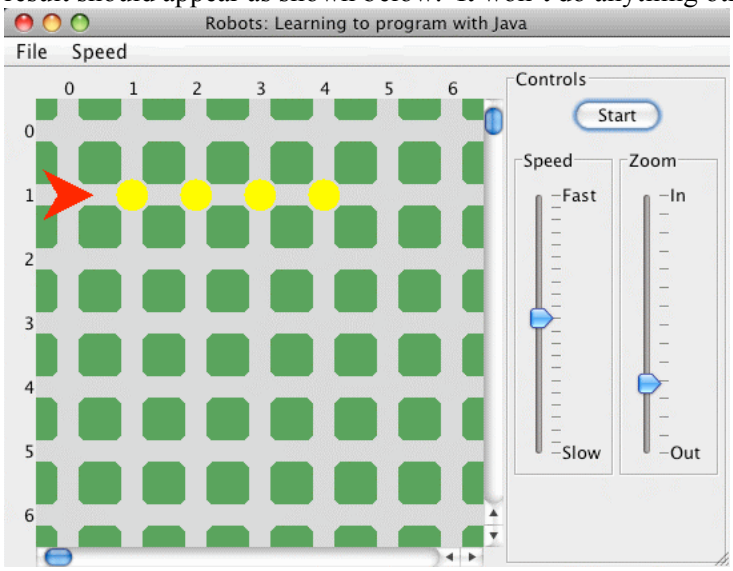
```
import becker.robots.*;

public class PickFour extends Object
{
  public static void main(String[] args)
  {
    City waterloo = new City("pickFour.txt");
    Robot karel = new Robot(waterloo, 1, 0, Direction.EAST);

    // Instructions to karel the robot go here.
  }
}
```

Hands On:

- Download the workshop materials:
  - Start the Safari web browser
  - Go to www.cs.uwaterloo.ca/~bwbecker/BeginJava.zip It should download to your desktop and unzip itself.
- Open the file `PickFour.java` in an editor. It should look like the above.
- Compile it:
  - Stuff to do each time you log in:

    - open an X terminal to use for compiling and running the programs:
    - type `cd Desktop/BeginJava` to change directories into the directory we'll use for this workshop
    - type `setenv CLASSPATH becker.java:.` (include the trailing colon and period). This tells the compiler to use the `becker.jar` library.
  - Stuff to do every time you want to compile a program:
    - type `javac «ClassName».java` (for example, `javac PickFour.java`). This runs the compiler to convert the source code into byte code

- type `java «ClassName»` (for example, `java PickFour`) to run the program. The result should appear as shown below. It won't do anything other than display itself (yet).



- Modify the program so the robot will pick up all four things and stop on Avenue 5. Compile and run the program to verify your solution.
- Create a new program, `PickFour2.java`. You will need to change `PickFour` in line 3 to `PickFour2`. Modify this program to use two robots to pick up the things, one starting on Avenue 0 and another starting on Avenue 5. Each robot should pick up two of the things.

Discussion

What Else Can Robots Do?

http://learningwithrobots.com/doc/becker/robots/Robot.html

Commands:

- void move()
- void pickThing()
- void putThing()
- void setSpeed(double movesPerSecond)
- void turnLeft()

Queries:

- boolean canPickThing()
- int countThingsInBackpack()
- boolean frontIsClear()
- int getAvenue()
- Direction getDirection()
- double getSpeed()
- int getStreet()

Control Structures

IF Statement

```
if («boolean test»)
{ «statement list»
}
```

```
if (karel.frontIsClear())
{ karel.move();
}
```

```
if («boolean test»)
{ «statement list»
} else
{ «statement list»
}
```

```
if (karel.frontIsClear())
{ karel.move();
} else
} karel.turnLeft();
}
```

```
if («boolean test»)
{ «statement list»
} else if («boolean test»)
{ «statement list»
} ...
```

WHILE Statement

```
while («boolean test»)
{ «statement list»
}
```

```
while (karel.canPickThing())
{ karel.pickThing();
}
```

FOR Statement

```
for («init»; «test»; «update»)
{ «statement list»
}
```

```
for (int i=0; i<4; i++)
{ karel.pickThing();
}
```

equivalent `while` loop:

```
«init»;
while («test»)
{ «statement list»
  «update»;
}
```

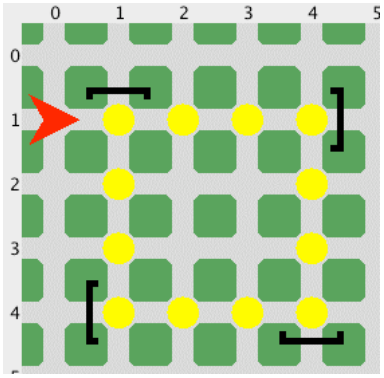
Boolean Expressions

- not: `!«boolExpr»`
- and: `«boolExpr1» && «boolExpr2»` (`boolExpr2` is evaluated only if necessary)
- or: `«boolExpr1» || «boolExpr2»` (`boolExpr2` is evaluated only if necessary)

Hands On

- Open **PickFive.java**. It has five thing objects lined up in front of a wall. Use a loop to pick up all five things.

- Open **PickSquare.java**. Pick up all the things, leaving the robot facing north on (1, 1).



Inheritance

Robots are pretty limited in what they can do. For example, it would be nice to have a kind of robot that could turn right with a single command instead of using **turnLeft** three times. A class can be *extended* to do new things as shown:

```
import becker.robots.*;

public class MyBot extends Robot
{
  public MyBot(City c, int str, int ave, Direction dir)
  {
    super(c, str, ave, dir);
  }

  public void turnRight()
  { this.turnLeft();
    this.turnLeft();
    this.turnLeft();
  }
}
```

A program that uses it:

```
import becker.robots.*;

public class PickSquare2 extends Object
{
  public static void main(String[] args)
  { City waterloo = new City("pickSquare.txt");
    MyBot karel = new MyBot(waterloo, 1, 0, Direction.EAST);

    // Instructions to karel the robot go here.
    karel.turnRight();
  }
}
```

Hands On

- Think about what additional capabilities a robot should have to make your life as a `PickSquare` programmer easier (but not trivial). Add those capabilities to `MyBot.java`. Modify `PickSquare2.java` to pick up all the things, as before, but using your new kind of robot.
- To compile, either use `javac` separately on each file or both at the same time
  `javac PickSquare2.java MyBot.java`
  `java PickSquare2`

Writing a Class from Scratch

An object like a robot has a bunch of information it needs to keep about itself. For example, open and run `Vars.java`:

```
import becker.robots.*;

public class Vars extends Object
{
  public static void main(String[] args)
  {
    City waterloo = new City("pickSquare.txt");
    Robot karel = new Robot(waterloo, 1, 0, Direction.EAST);
    Robot sue = new Robot(waterloo, 5, 1, Direction.NORTH);

    karel.setSpeed(0.5);
    while (karel.frontIsClear())
    { karel.move();
      karel.pickThing();
    }

    System.out.println("\t\tKarel\tSue");
    System.out.println("Avenue:\t\t" + karel.getAvenue() + "\t" + sue.getAvenue());
    System.out.println("Street:\t\t" + karel.getStreet() + "\t" + sue.getStreet());
    System.out.println("Direction:\t" + karel.getDirection()
                       + "\t" + sue.getDirection());
    System.out.println("Num Things:\t" + karel.countThingsInBackpack()
                       + "\t" + sue.countThingsInBackpack());
    System.out.println("Speed:\t\t" + karel.getSpeed() + "\t" + sue.getSpeed());
  }
}
```

Look at the terminal window to see the output.

You can think of the `Robot` class as being similar to the following (this is much simplified, of course):

```
public class Robot extends Object
{
  private int avenue;
  private int street;
  private Directon dir = Direction.EAST;
  private double speed = 2.0;
  // etc

  public Robot(City c, int str, int ave, Direction d)
  { super();
    this.street = str;
    this.avenue = ave;
    this.dir = d;
  }

  public void move()
  { if (this.dir == Direction.EAST)        this.ave = this.ave + 1;
    else if (this.dir == Direction.WEST)   this.ave = this.ave - 1;
    else if (this.dir == Direction.SOUTH)  this.str = this.str + 1;
    else if (this.dir == Direction.NORTH)  this.str = this.str - 1;
    else throw new Error("Oops... invalid direction!");
  }
}
```

Hands On

- Open `Account.java` and `Bank.java`. Leave `Bank.java` alone; modify `Account.java` to model a bank account for use by the bank.
- Test your solution with the following commands:
  ```
  javac Bank.java Account.java
  java -ea Bank
  ```
  The `-ea` turns on assertions. They can be useful for checking (testing!) that things you know ought to be true really are true.
- Suggestion: work incrementally. Do what you need to do to get the uncommented code working. Then work on getting the next assertion to pass, etc.