RobotWorld

Introduction to RobotWorld

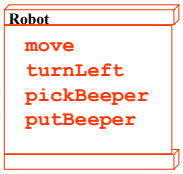
An environment to teach Java programming

Overview

- The Robot class
- Robot methods
  - move, turnLeft, pickBeeper, putBeeper
- An example Robot program
- Components of a Java program
  - Semicolons, method calls, reserved words
- Indentation
- Errors

The Robot Class

- This is the class that can be used to create *instances* of actual Robots in the robot world
- It has four methods:
  - » move
  - » turnLeft
  - » pickBeeper
  - » putBeeper



move

- When a robot executes a *move* instruction, it moves forward one block
- It continues to face the same direction
- The robot will not move forward if obstructed by a wall section or boundary wall, instead an *error* occurs and the program stops.

turnLeft

- A robot executes *turnLeft* by pivoting 90 degrees to the left
- A robot stays on the same cell
- A wall cannot block a robot's turn, so *turnLeft* cannot cause an error

move and turnLeft

- A robot always starts in a cell, facing north, south, east or west
- He can't move fractions of a cell or turn other than 90 degrees, so after *move* or *turnLeft* he is still in a cell and still facing north, south, east or west
- What about "turnRight", which doesn't exist as a primitive instruction?

pickBeeper

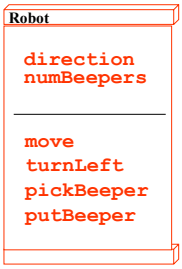
- A robot executes a *pickBeeper* by picking up a beeper from the cell he is on and putting it in his bag
- If there is no beeper there, an error occurs
- If there is more than one beeper, he randomly picks up one and puts it in the bag
- beepers are small; they never block movement

putBeeper

- A robot executes *putBeeper* by taking a beeper from the bag and putting it on current cell
- If beeper bag is empty, an *error* occurs

The Robot Class

- This class has not only four methods, but also two attributes
  - » direction
  - » numBeepers
- Each has the obvious meaning



An Instance of Robot

- So let's say we want to create an instance of Robot, called bill, standing at cell (0, 0), facing East
- We write

```
Robot bill = new Robot();
```

- Now, we add bill to a world

```
world.add(bill);
```

That is, we use the *add* method of world.

An example program

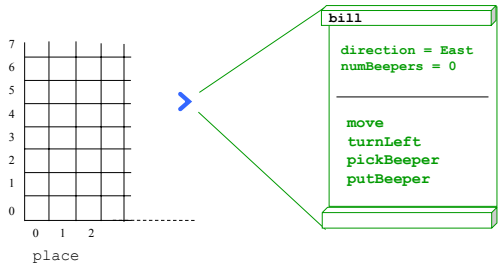
```
public class Test
{
   public static void main(String [] args)
   {
      World place = new World();
      Robot bill = new Robot();
      place.add(bill);
   }
}
```

Create an instance of a World, call it place

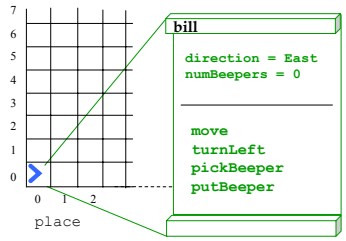
Create an instance of a Robot, call it bill

Add bill to the world by calling the add method of place.

```
World place = new World();
Robot bill = new Robot();
place.add(bill);
```



```
World place = new World();
Robot bill = new Robot();
place.add(bill);
```



Primitive Instructions and Simple Programs

- We get a robot to *execute* an instruction by invoking (calling) the appropriate method.
- We *execute* a program by executing each instruction in the program

A Complete Program

Here is a problem.

We want a robot, starting at cell (0, 0), to go to cell (3, 1), pick up a beeper and bring it to cell (3, 3).

After depositing the beeper, the robot should move one block farther north.

```
public class BeepMove
{
   public static void main(String [] args)
   {
      World place = new World();
      place.addBeeper(3, 1);
      Robot bill = new Robot();
      place.add(bill);
      bill.move();
      bill.move();
      bill.move();
      bill.turnLeft();
      bill.move();
      bill.pickBeeper();
      bill.move();
      bill.putBeeper();
      bill.move();
   }
}
```

Executing a program

- The program sequentially executes each instruction in the main() method, in strict top to bottom order
  - Creates the world
  - Creates the Robot
  - Adds the robot to the world
  - The instructions are executed until the end, unless an error occurs

Simulating a Program

Simulating a program means that we execute the program exactly as it would on the computer, recording each action that takes place

We can hand-simulate a robot program on paper

The ability to simulate robot behavior is an important skill we must acquire.

The Form of a Java Program

The symbols of a Java program are divided into three classes:

- Punctuation Marks, e.g. the semicolon **;**
- Methods, e.g. move(), addWall() etc.
- Reserved Words

Reserved Words

- We have already seen reserved words in our program: *public*, *class* and *new*

```
public class Test
{
   public static void main(String [] args)
   {
      World place = new World();
      Robot bill = new Robot();
      place.add(bill);
   }
}
```

Semicolons and Braces

Semicolons terminate instructions

```
move();
turnLeft();
```

Braces are used to contain a class or method definition

```
public class Test
{
   public static void main(String [] args)
   {
      World place = new World();
   }
}
```

Indentation

- Indentation makes programs easier to read, so get in the habit of indenting your programs correctly.
- The part that braces surround are indented (i.e. moved over to the right).

```
public class Test
{
   public static void main(String [] args)
   {
      World place = new World();
      Robot bill = new Robot();
      place.add(bill);
   }
}
```

Indentation

- The following is not correctly indented:

```
public class Test
{
public static void main(String [] args)
{
World place = new World();
Robot bill = new Robot();
place.add(bill);
}
}
```

Whatever the braces enclose should be indented

Indentation

- Move this code over by three spaces

```
public class Test
{
public static void main(String [] args)
{
World place = new World();
Robot bill = new Robot();
place.add(bill);
}
}
```

Indentation

- Move this code over by three spaces

```
public class Test
{
   public static void main(String [] args)
   {
   World place = new World();
   Robot bill = new Robot();
   place.add(bill);
   }
}
```

Indentation

- The inner set of braces should also have its contents indented

```
public class Test
{
   public static void main(String [] args)
   {
   World place = new World();
   Robot bill = new Robot();
   place.add(bill);
   }
}
```

Whatever the braces enclose should be indented

Indentation

- The inner set of braces should also have its contents indented

```
public class Test
{
   public static void main(String [] args)
   {
      World place = new World();
      Robot bill = new Robot();
      place.add(bill);
   }
}
```

This program is now correctly indented.

Errors

Errors can occur after

- a *move* method
- a *pickBeeper* method
- a *putBeeper* method

Errors

To avoid errors:

- Execute a move() method, only when the robot's path is clear
- Execute a pickBeeper() method only when there is at least one beeper on the cell.
- Execute a putBeeper() method only when the beeper-bag is not empty

Programming Errors

Programming errors can be classified into three broad categories:

- Syntactic Errors
- Execution Errors
- Logic Errors

Syntactic Errors

A syntactic error occurs when we use incorrect grammar or punctuation:

Syntactic Error, missing first {

```
public class Test

    public static void main(String [] args)
    {
        World place = new World();
    }
}
```

Execution Errors

Execution errors occur when the robot is unable to execute a method successfully and an error occurs:

```
public static void main(String [] args)
{
    World garden = new World();
    Robot jill = new Robot();

    garden.add(jill);

    jill.turnLeft();
    jill.turnLeft();
    jill.move();
}
```

An Execution Error will occur (crash into the wall)

Logic Errors

The worst of all. The robot successfully completes the program, but not the task.

A logic error may occur early in a program and lead to an execution error later on. Then, we must trace backward from the instruction that caused an error, to find the instruction that started the robot in the wrong direction.

Logic Error

Move *jill* forward three squares, turn left and moves forward one square

The final *move* method was forgotten; the task was not completed.

```
jill.move();
jill.move();
jill.move();
jill.turnLeft();
```

Bugs and Debugging

- All types of errors are known as “bugs”
- Debugging means removing errors from a program
- We’ll learn a lot about debugging…

Summary

- The World and Robots are objects.
- Robots have some useful methods, e.g. move, turnleft, putBeeper and pickBeeper.
- You can program the robot to solve certain tasks.
- Indent your programs to show the structure.
- You must be careful of errors in your program.

Addendum

- Can there be more than one Robot in a World?
- Yes, Robot defines the class, but there can be many instances of Robot.
- However:
  - they would have to be given a different name
  - they would have to be placed in different cells, use the add(x, y) method.