Decomposition

Chris Piech

CS106A, Stanford University
Today’s Goal

1. Be able to approach a problem “top down” by using decomposition (aka top town refinement)
First, a cool program
Quick review
Karel the Robot

Piech, CS106A, Stanford University
import stanford.karel.*;

public class MethodExample extends SuperKarel {

    public void run() {
        goToMoon();
    }

    private void goToMoon() {
        buildSpaceship();
        // a few more steps
    }

    private void buildSpaceship() {
        // todo
    }
}
```java
import stanford.karel.*;

public class ForExample extends SuperKarel {

    public void run() {
        // repeats the body 99 times
        for(int i = 0; i < 99; i++) {
            // the "body"
            putBeeper();
        }
    }
}
```
While Loops

```java
import stanford.karel.*;

public class WhileExample extends SuperKarel {

    public void run() {
        // while condition holds runs body
        // checks condition after body completes
        while (frontIsClear()) {
            move();
        }
    }
}
```
import stanford.karel.*;

public class IfExample extends SuperKarel{

    public void run() {

        // If the condition holds, runs body
        if(frontIsClear()) {
            move();
        }
    }
}

Piech, CS106A, Stanford University
import stanford.karel.*;

public class IfExample extends SuperKarel{

   public void run() {
      // If the condition holds,
      if(beepersPresent()) {
         // do this
         pickBeeper();
      } else {
         // otherwise, do this
         putBeeper();
      }
   }
}

If / Else Statement
The Full Karel

Today we will use:
- `frontIsClear()`
- `rightIsBlocked()`
- `beepersPresent()`

Built-in Karel commands:
- `move()`
- `turnLeft()`
- `putBeeper()`
- `pickBeeper()`

Karel program structure:
```java
import stanford.karel.*;

public class name extends Karel {
    public void run() {
        statements in the body of the method
    }
}
```

Karel condition names:
- `frontIsClear()`
- `frontIsBlocked()`
- `leftIsClear()`
- `leftIsBlocked()`
- `rightIsClear()`
- `rightIsBlocked()`
- `beepersPresent()`
- `noBeepersPresent()`
- `beepersInBag()`
- `noBeepersInBag()`
- `facingNorth()`
- `notFacingNorth()`
- `facingEast()`
- `notFacingEast()`
- `facingSouth()`
- `notFacingSouth()`
- `facingWest()`
- `notFacingWest()`

Conditional statements:
```java
if (condition) {
    statements executed if condition is true
}
else {
    statements executed if condition is false
}
```

Iterative statements:
```java
for (int i = 0; i < count; i++) {
    statements to be repeated
}
while (condition) {
    statements to be repeated
}
```

Method definition:
```java
private void name () {
    statements in the method body
}
```

New commands in the SuperKarel class:
- `turnRight()`
- `turnAround()`
- `paintCorner(color)`

New conditions in the SuperKarel class:
- `random()`
- `random(p)`
- `cornerColorIs(color)`
You might want to combine conditions:

This means “and”

```java
if(frontIsClear() && rightIs Blocked()) {
    ... some code
}
```

This means “or”

```java
if(frontIsClear() || rightIsClear()) {
    ... some code
}
```
End review
private void friday(){
    banishWinter();
    decomposition();
    doubleBeepers();
    rhoombaKarel();
    if(extraTime()){
        wordSearchKarel();
    }
}
Banish Winter
private void friday(){
    banishWinter();
    decomposition();
    doubleBeepers();
    rhoombaKarel();
    if(extraTime()){
        wordSearchKarel();
    }
}
private void friday(){
    banishWinter();
    decomposition();
    doubleBeepers();
    rhoombaKarel();
    if(extraTime()){
        wordSearchKarel();
    }
}
private void friday(){
    banishWinter();
    decomposition();
    doubleBeepers();
    rhoombaKarel();
    if(extraTime()){
        wordSearchKarel();
    }
}
Double Beepers

Before

After

Muhammed ibn Musa Al Kwarizmi
Do your thing
Aside: Common Errors

Lather, Rinse, Repeat
private void turnToWall() {
    while (leftIsClear()) {
        turnLeft();
    }
}
Infinite Loop

```java
private void turnToWall() {
    while(leftIsClear()) {
        turnLeft();
    }
}
```
private void turnToWall() {
    while(leftIsClear()) {
        turnLeft();
    }
}
private void turnToWall() {
    while (leftIsClear()) {
        turnLeft();
    }
}
Pre/Post that Don’t Match

```java
private void addLeavesToTrees() {
    turnLeft();
    climbTree();
    addLeaves();
    descendToGround();
    turnLeft();
}
```

* Pro tip: I also think of while loops as having a pre and post condition, which must match
private void friday() {
    banishWinter();
    decomposition();
    doubleBeepers();
    rhoombaKarel();
    if (extraTime()) {
        wordSearchKarel();
    }
}
private void friday() {
    banishWinter();
    decomposition();
    doubleBeepers();
    rhoombaKarel();
    if (extraTime()) {
        wordSearchKarel();
    }
}


• Write a Roomba Karel that sweeps the entire world of all beepers.
  – Karel starts at (1,1) facing East.
  – The world is rectangular, and some squares contain beepers.
  – There are no interior walls.
  – When the program is done, the world should contain 0 beepers.
  – Karel's ending location does not matter.

• How should we approach this tricky problem?
Possible Algorithm 1
Possible Algorithm 2

![Possible Algorithm Diagram]
Possible Algorithm 3
Possible Algorithm 4
Rhoombomba Karel
private void friday(){
    banishWinter();
    decomposition();
    doubleBeepers();
    rhoombaKarel();
    if(extraTime()){
        wordSearchKarel();
    }
}
private void friday(){
    banishWinter();
    decomposition();
    doubleBeepers();
    rhoombaKarel();
    if(extraTime()){ 
        wordSearchKarel();
    }
}
private void friday()
{
    banishWinter();
    decomposition();
    doubleBeepers();
    rhoombaKarel();
    if(extraTime()){
        wordSearchKarel();
    }
}
Happy Friday