Software Engineering

January 10, 2006

Agenda

- **Getting Started**
- Design Principles
- Threading in Java

The Design Sequence

- Open a text editor to edit a source code file:
  - `emacs MyExample.java`
- Write class declaration, and declarations for each of your methods, annotated with comments.
- Fill in source code.
- Compile:
  - `javac MyExample.java`
    - This produces `MyExample.class` if successful
- Fix compile errors and repeat compilation until successful.
- Run and debug behaviors:
  - `java MyExample`
    - This searches the `CLASSPATH` for `MyExample.class` and executes it.
- Back up files.

The Design Sequence

- Open a text editor to edit a source code file:
  - `emacs MyExample.java`
- Write class declaration, and declarations for each of your methods, and with comments.
- Put these lines in your Athena `~/.environment`:

```
add 6.186
add -f java_v1.5.0
setenv JAVA_HOME /mit/java_v1.5.0
setenv CLASSPATH /mit/6.186/2006/maslab.jar:.
```

- Adjust your `~/.emacs`
  - This searches the `CLASSPATH` for `MyExample.class` and executes it.
- Back up files.

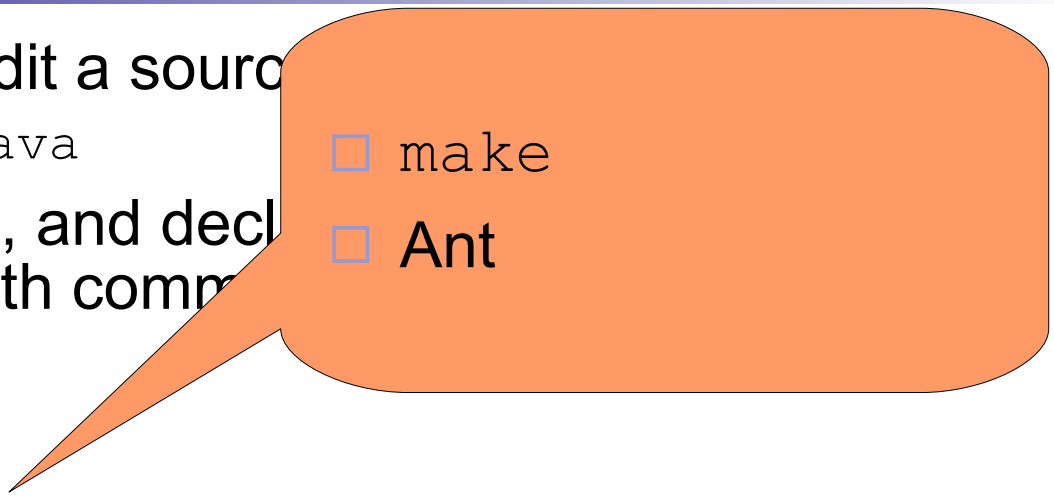
The Design Sequence

- Open a text editor to edit a source code file:
  - `emacs MyExample.java`
- Write class declaration, and declarations for each of your methods, annotated with comments.
- Fill in source code.
- Compile:
  - `javac MyExample.j`
    - This produces `MyExam`
- Fix compile errors and
successful.
- Run and debug behavi
  - `java MyExample`
    - This searches the `CLASSPATH` for `MyExample.class` and executes it.
- Back up files.

- Java 1.5.0 API
- APIs for Maslab, OrcBoard, Jama
- Sun's Java Tutorial
- Ed Faulkner's Java Reference

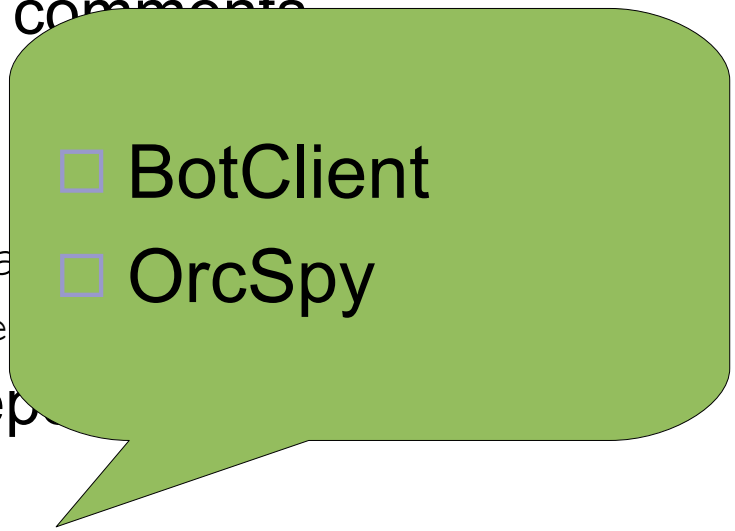
The Design Sequence

- Open a text editor to edit a sourc
  - `emacs MyExample.java`
- Write class declaration, and decl
  methods, annotated with comm
- Fill in source code.
- Compile:
  - `javac MyExample.java`
    - This produces `MyExample.class` if successful
- Fix compile errors and repeat compilation until successful.
- Run and debug behaviors:
  - `java MyExample`
    - This searches the `CLASSPATH` for `MyExample.class` and executes it.
- Back up files.



The Design Sequence

- Open a text editor to edit a source code file:
  - `emacs MyExample.java`
- Write class declaration, and declarations for each of your methods, annotated with comments.
- Fill in source code.
- Compile:
  - `javac MyExample.java`
    - This produces `MyExample`
- Fix compile errors and rep successful.
- Run and debug behaviors:
  - `java MyExample`
    - This searches the `CLASSPATH` for `MyExample.class` and executes it.
- Back up files.



The Design Sequence

- Open a text editor to edit a source code file:
  - `emacs MyExample.java`
- Write class declaration, and declarations for each of your methods, annotated with comments.
- Fill in source code.
- Compile:
  - `javac M`
    - This p
- Fix compil
  successfu
- Run and c
  - `java MyExam`
    - This search the `CLASSPATH` for `MyExample.class` and executes it.
- Back up files.
  - CVS
  - Subversion
  - All tools described in the software section of the wiki

Agenda

- Getting Started
- **Design Principles**
- Threading in Java

Design Principles: Motivation

Coding a Maslab Robot is a formidable, multi-person project. **Good design** is critical!

- Makes debugging easier.
- Makes sure different team members' code all works together.
- Makes sure one team member's changes doesn't break another team member's code.

General Tips for Maslab

- Have a software design plan before you begin coding.
- Always maintain a working version of your code as you improve and replace modules.
- Discuss coding style and CVS/Subversion etiquette with your team members (i.e., do not check in non-functional code, etc.).

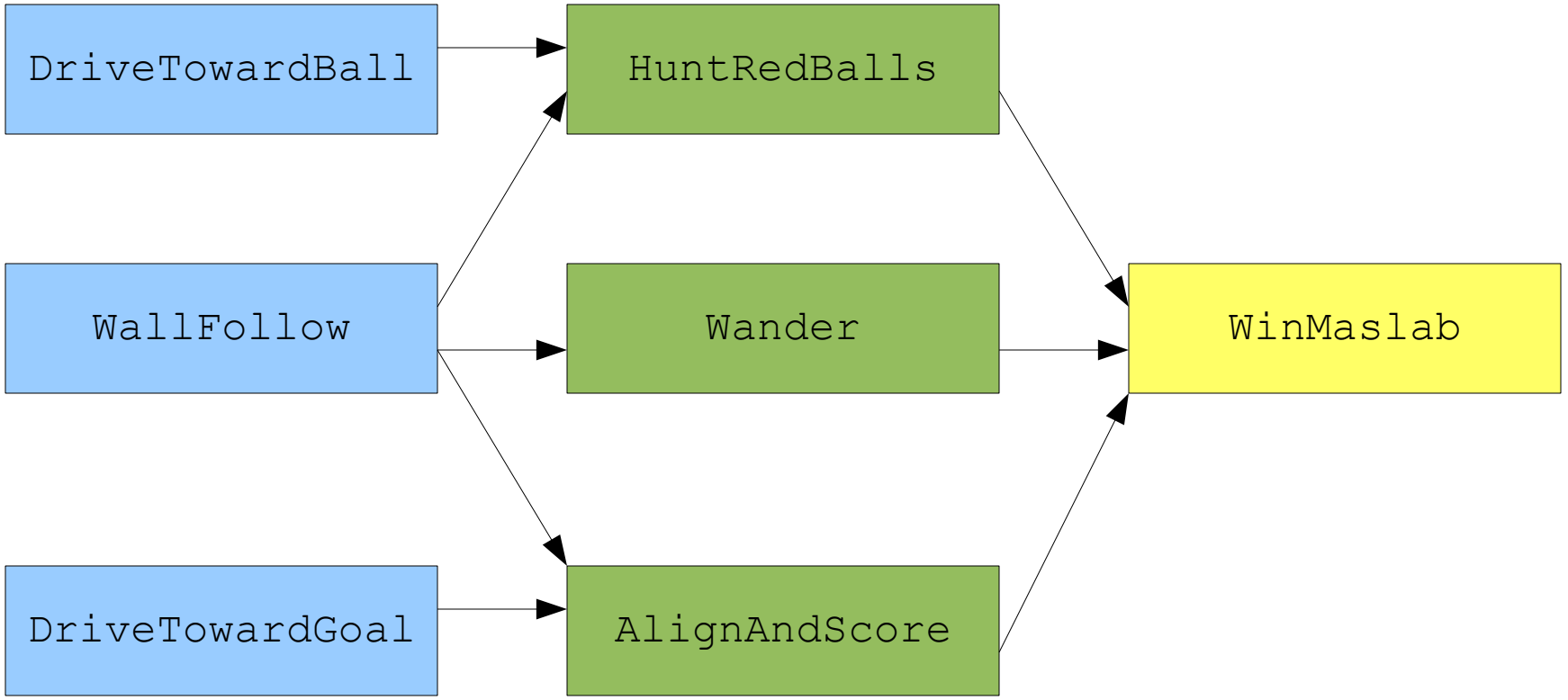
Good Design Practices

- Thou Shalt Test Constantly.
- Start small, build up.
- Modularity.
- Avoid over-abstraction.
- Back up code.
  - Keep multiple versions backed up.
  - Keep separate backups off of the robot computer.

The Design Process

- Top-down vs. Bottom-up (we suggest bottom-up).
- Write out specifications for each module.
- Write code for modules.
- Test each module separately as it is being written.
- Test overall system for functionality.

Modular Design



Testing

- Test each module separately.
- Test overall system.
- Test special cases.
- Come up with test cases before coding, or have a different team member do testing.
- Use the `main()` method.
- Unit testing.

Agenda

- Getting Started
- Design Principles
- **Threading in Java**

Motivation for Threading

- Ability to perform tasks in parallel.
- If used properly, threading can make your robot run faster.
- Different threads for:
  - Image capture and processing
  - Keeping a current map
  - Controlling the current motion behavior (wandering, ball-seeking, obstacle avoidance, etc.)
  - Higher-level strategic control

Using Threading

- Look in Sun's Java tutorial, or Ed Faulkner's Java reference.
- Look at the Java API:
  - `Thread`, `Runnable`, `wait()`, `notify()`, `sleep()`, `yield()`
- Take care to avoid deadlock.

Synchronization in Threading

- Using the same object from two threads at the same time can cause your program to break.
- `synchronize` allows blocks of code to be mutually exclusive.

Threading Example

```
public class Odometry {
  public class Position {
    public double x, y, th;
    public Position(x, y, th) { ... }
  }
  Position pos;
  public static void main(String[] args) {
    Orc orc = Orc.makeOrc();
    Odometry od = new Odometry(orc);
    while (true) {
      Position pos = od.pos;
      System.out.println("(" + pos.x + "," + pos.y + ") " + pos.th);
      Thread.sleep(250);
    }
  }
  QuadPhase encL, encR;
  public Odometry(Orc orc) {
    encL = new QuadPhase(orc, 0); encR = new QuadPhase(orc, 1);
    OdometryThread odthread = new OdometryThread();
    odthread.start();
  }
  ...
```

Threading Example

```
public class Odometry {
  ...
  class OdometryThread extends Thread {
    int lastCountL, lastCountR;
    void OdometryThread() {
      lastCountL = encL.getCount(); lastcountR = encR.getCount();
    }
    public void run() {
      while (true) {
        int countL = encL.getCount(), countR = encR.getCount();
        double distL = (countL - lastCountL) * speed,
               distR = (countR - lastCountR) * speed;
        lastCountL = countL; lastCountR = countR;
        double dist = (distL + distR) / 2.,
               rot = (distR - distL) / 2. / wheelDist;
        pos = new Position(pos.x + dist*Math.cos(pos.th + rot/2.),
                           pos.y + dist*Math.sin(pos.th + rot/2.),
                           pos.th + rot);
        Thread.sleep(20);
      }
    }
  }
}
```