



**Game Design in Object-Oriented Style:  
Data, Tests, Programs**

**Viera Krňanová Proulx**

**Northeastern University**

**[vkp@ccs.neu.edu](mailto:vkp@ccs.neu.edu)**



## ➤ **What is Introductory Computing?**

- Introductory Computing Options
- Programming: Why and How

## ➤ **Programming and Design**

## ➤ **Pedagogical Innovations**

## ➤ **Our Experiences**

# Introduction



## Introductory Computing Options

- Programming as a game: Logo, Alice, Scratch
- Learning to use applications: Word, Excel, Database, Web
- Programming robots: Lego Mindstorms, Cricket
- Web design, multimedia
- Programming as a computation -- algorithmics
- Language-driven programming
- Programming as design -- data centric

# Introduction



## Introductory Computing Options

Programming as a game: Logo, Alice, Scratch

<http://www.logo.com/imagine>

The screenshot shows a web browser window titled "Logotron :: Imagine - Gallery". The address bar shows the URL "http://www.logo.com/imagine/gallery.html". The browser's search bar contains the name "Adrion". The website header features the Logotron logo (a turtle) and the text "Logotron educational software" with the tagline "Partners with the teaching profession - Pioneers in Learning". A navigation menu includes links for "home", "about", "products", "support", "my cart", and "search".

The main content area is titled "Imagine Logo" and "Enabling learners to do more by doing less...". It features a "Gallery of Work - 1" section with the following text:

*Here you will find two pages of some of the best projects created by Imagine Logo users. If you produce something interesting that you think others would like to see, please let us know by emailing [support@logo.com](mailto:support@logo.com) with the message title Imagine Logo Gallery.*

You need to have a suitably configured web browser, and the Imagine Logo web plugin installed.

Internet Explorer, Mozilla Firefox and Opera web browsers work with the Imagine Logo plugin. You need to be running a Windows based operating system (98,2000 or XP) too.

Enjoy...

Two project thumbnails are shown: "Level 2" (a grid-based game) and "Spiral World" (a geometric spiral).

On the right side, there is a "buy online now!" button and a "MORE INFO" section with the following links:

- Download FREE demo
- Calling all First Logo, WinLogo & SuperLogo Users
- Use Creative Classroom Online
- Buy Creative Classroom
- Step by Step Activities
- Twenty Things to Do with a Computer - by Seymour Papert & Cynthia Solomon
- Design & evaluation of Maths related programs for Special Education
- Imagine Logo Workbooks
- Logopoint

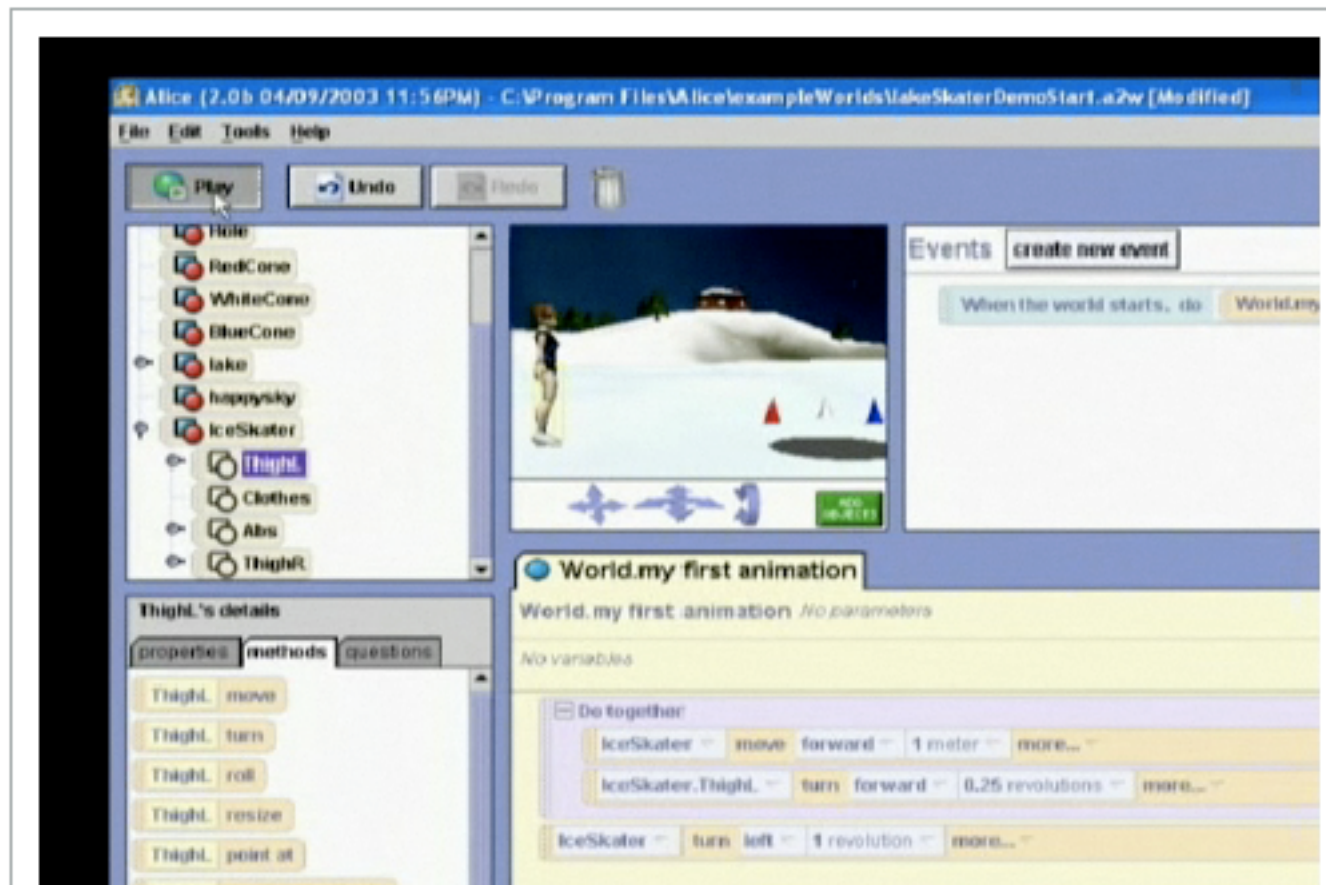
# Introduction



## Introductory Computing Options

Programming as a game: Logo, Alice, Scratch

<http://www.alice.org>



# Introduction



## Introductory Computing Options

Programming as a game: Logo, Alice, Scratch

<http://scratch.mit.edu>

The screenshot shows the Scratch website homepage. At the top, there is a navigation bar with links for home, projects, galleries, support, forums, about, and my stuff. Below this is a search bar and a login/signup prompt. The main content area is divided into several sections: a central banner with Scratch blocks and the Scratch cat, a 'Download Scratch' section, 'Newest Projects' featuring 'Crazy Jet!!!!', 'Driving Simulator', and 'Block party!', 'Featured Projects' featuring 'Mimi - part 11', 'DoYouKnow The States...', and 'Graffiti', a 'Scratch@MIT Conference' announcement, 'Featured Galleries' with 'Cyclone103s Fa...', 'Seeking Feedba...', and 'scratch kat pr...', and a 'Scratch Club' section with 'Scratch Story Projects'.

# Introduction



## Introductory Computing Options

### Programming as a game: Logo, Alice, Scratch

- **Advantage:** nice images - excitement
- **Advantage:** tinkering - exploration
- **Disadvantage:** tinkering, not planning
- **Disadvantage:** (except Logo) - no way of composing small parts into bigger ones
- **Disadvantage:** where do we go from here?

# Introduction



## Introductory Computing Options

Learning to use applications: Word, Excel, Database, Web

[http://wps.aw.com/aw\\_snyder\\_fluency\\_1/](http://wps.aw.com/aw_snyder_fluency_1/)

The screenshot shows the top navigation bar with the Pearson Addison Wesley logo and the title 'fluency with Information Technology Skills, Concepts, & Capabilities'. Below the navigation bar are buttons for 'Home', 'Search', 'Help', and 'Profile'. The main content area features the title 'Addison Wesley Companion Website Fluency with Information Technology: Skills, Concepts, and Capabilities'. A sidebar on the left contains the author's name 'LAWRENCE SNYDER' and a smaller version of the book's logo. The main text area contains a welcome message and a link to 'Student Resources' with a description of the lab workbook and supporting files.

PEARSON  
Addison  
Wesley

*fluency* with Information Technology  
Skills, Concepts, & Capabilities

Inside: [Student Resources](#) [Home](#) [Search](#) [Help](#) [Profile](#)

[Home](#) >

**Addison Wesley Companion Website**  
**Fluency with Information Technology: Skills, Concepts, and Capabilities**

LAWRENCE SNYDER  
*fluency* with Information Technology  
Concepts, Capabilities, & Skills

Welcome to the Companion Web site for *Fluency with Information Technology: Cor Capabilities, and Skills* by Lawrence Snyder.

**Student Resources**  
Lab workbook, and supporting files from the book.



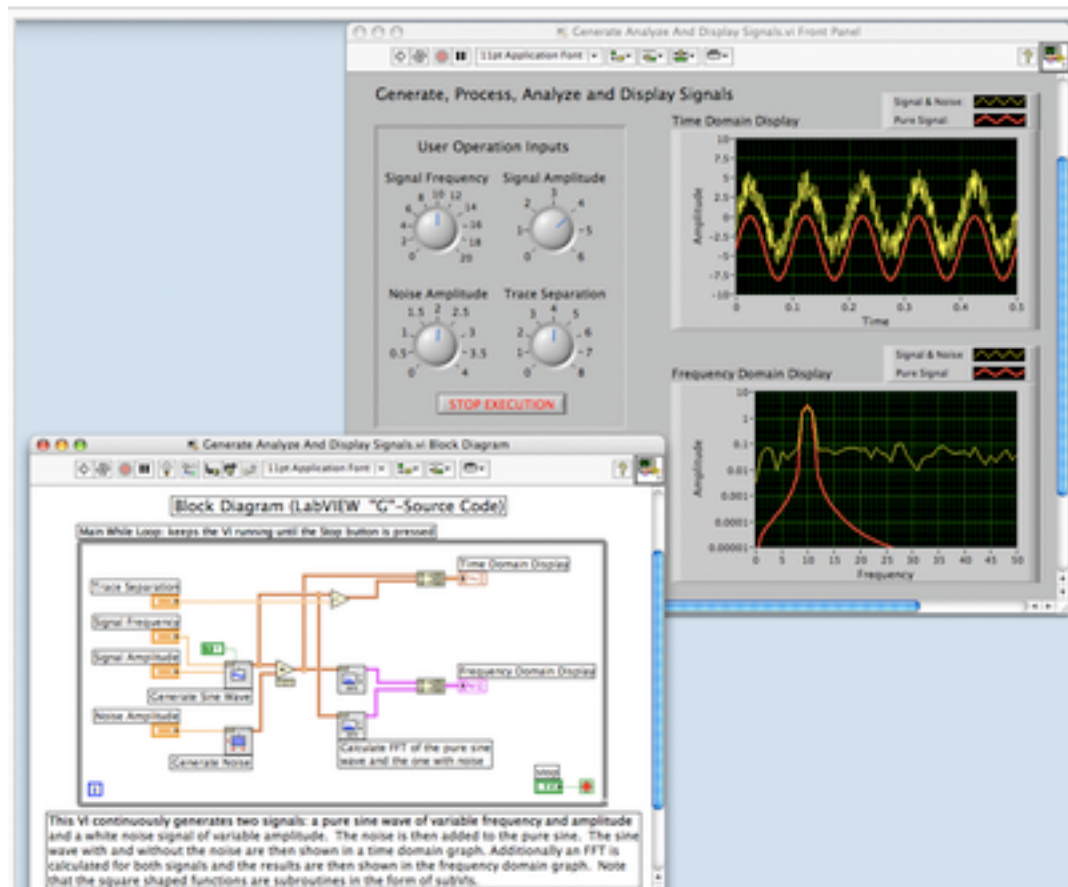
# Introduction



## Introductory Computing Options

Learning to use applications: Word, Excel, Database, Web

<http://LabView.com>



# Introduction



## Introductory Computing Options

Learning to use applications: Word, Excel, Database, Web

- **Advantage:** useful skills
- **Advantage:** some understanding of computing (if done well)
- **Disadvantage:** skills, not concepts
- **Disadvantage:** not encouraging to design abstractions
- **Disadvantage:** where do we go from here?

# Introduction



## Introductory Computing Options

Programming robots: Lego Mindstorms, Cricket

<http://mindstorms.lego.com>

LEGO.com MINDSTORMS NXTLOG

http://mindstorms.lego.com/NXTLOG/default.aspx

Apple (101) Home Gmail Isaac Lila IMICT2007 IMICT-cd 213-fl07 213-sp08 390 java JPT JPT2.5.0 Amazon eBay Yahoo!

LEGO HOME PRODUCTS PLAY SHOP

CHANGE REGION

play SIGN IN Register Now

**mINDSTORMS** HOME COMMUNITY NEWS PRODUCTS

Home - NXTLOG

**NXTLOG** 2.0 RULES L.D.O. ABOUT PROJECTS NEWS HELP

Welcome!

[Log in](#)

[Register for free](#)

Search

Search

**WE WANT YOU**

TO SHARE YOUR NXT PROJECT!

**GET STARTED**

**Featured Projects**

**Crash Lab (with the fl...**

[RrangerRick](#)

**MINDSTORMS NXT NXT Bu...**

[NXTLOGster](#)

**Vestas Windmill (turb...**

[robotica008](#)

**Automatic Hand Washer**

[Bucksinn92](#)

**TP-Bot 2008**

[Dino\\_Martino](#)

**Latest Projects**

**Musicbot**

[gl09aac](#)

**Broom Bot**

[JoeB360](#)

**Rover Chassis**

[Murcielago53](#)

**The Amazing Ball Writ...**

[crazykid506](#)

**Advanced Rubberband S...**

[rero57](#)

**Top 5 Projects**

**ScanBot**

[RCX-Inventor](#)

**Quadruped 1**

[TheFoxy](#)

**Compact Tribot**

[NXTPenguin465](#)

**Quad Runner**

[bbonahoom](#)

**Valentines Box 2**

[MasterEngineer555](#)

NXTlog News Popular Tags

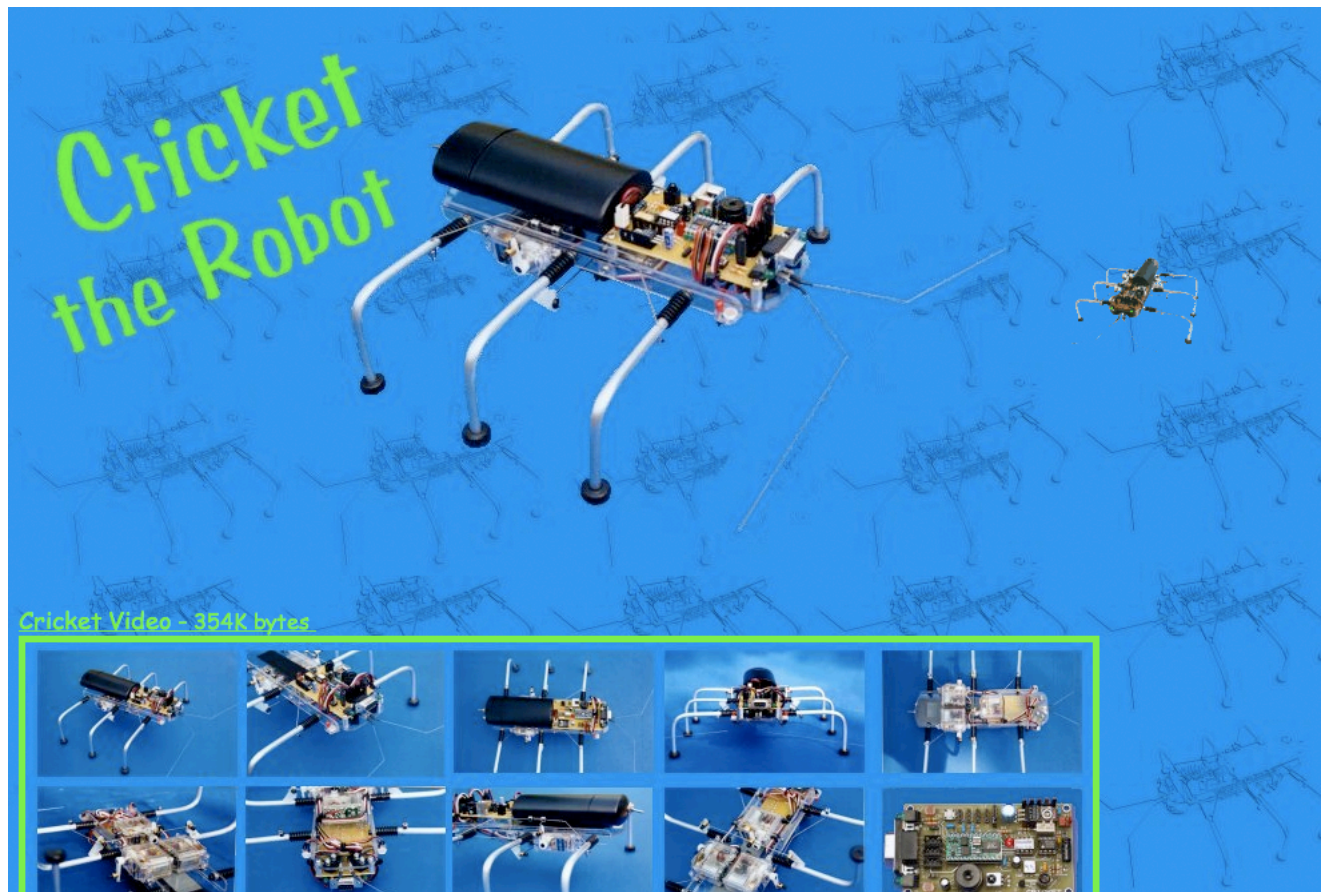
# Introduction



## Introductory Computing Options

Programming robots: Lego Mindstorms, Cricket

<http://www.cricket.org>



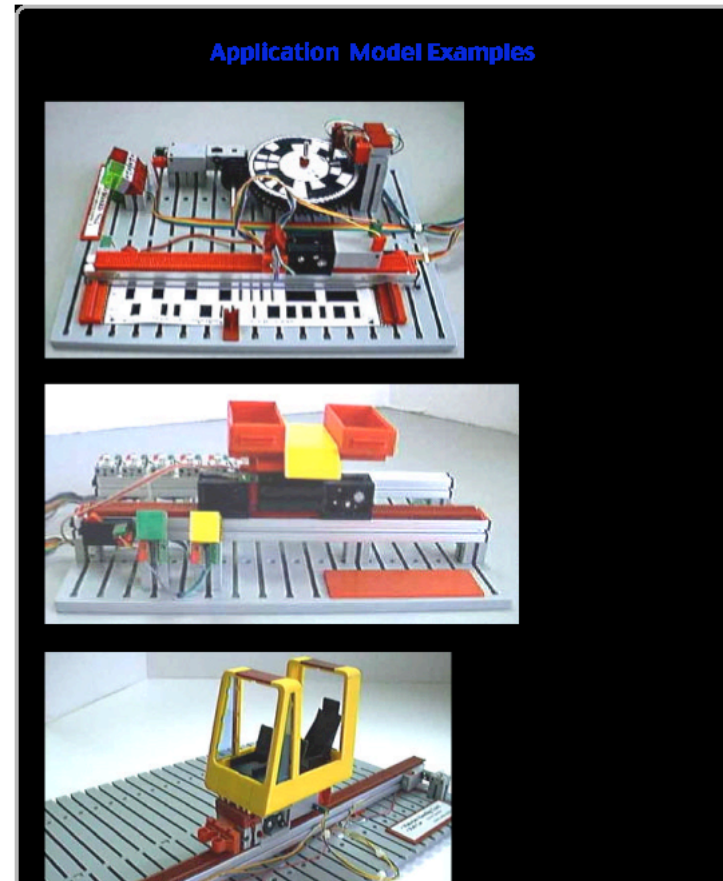
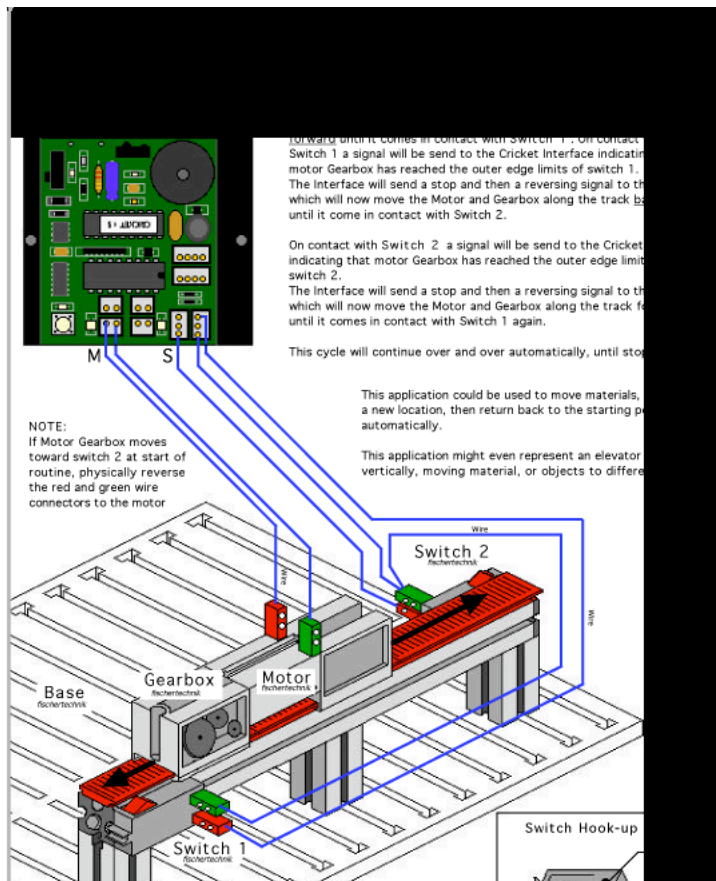
# Introduction



## Introductory Computing Options

Programming robots: Lego Mindstorms, Cricket

<http://www.cricket.org>



# Introduction



## Introductory Computing Options

Programming robots: Lego Mindstorms, Cricket

- **Advantage:** engaging - exploration
- **Advantage:** interesting environment
- **Disadvantage:** single mindset; specific details/skills
- **Disadvantage:** - no way of composing small parts into bigger ones
- **Disadvantage:** where do we go from here?

# Introduction



## Introductory Computing Options

### Web design, multimedia

- **Advantage:** exciting - attractive
- **Advantage:** useful, builds a community
- **Disadvantage:** skills - not concepts
- **Disadvantage:** abstractions? - maybe
- **Disadvantage:** where do we go from here?

# Introduction



## Introductory Computing Options

Programming as a computation -- algorithmics

```
public void quicksort(String source[]) {
    String pivot = source[0];
    partition(pivot, source, 0, Array.length(source));
}

public void partition(String pivot, String source[],
                    int low, int high) {
    for (int i = low; i < high; i++) {
        for j = high - 1; j > i; j--) {
            ...
        }
    }
}
```



# Introduction



## Introductory Computing Options

Programming as a computation -- algorithmics

- **Advantage:** serious thinking
- **Advantage:** important foundations
- **Disadvantage:** where do algorithms come from?
- **Disadvantage:** explicit design of abstractions?
- **Disadvantage:** how can we invent new ones?

# Introduction



## Introductory Computing Options

Language-driven programming

```
public static void main(String argv[]) {  
    System.out.println("Hello world");  
}
```

# Introduction



## Introductory Computing Options

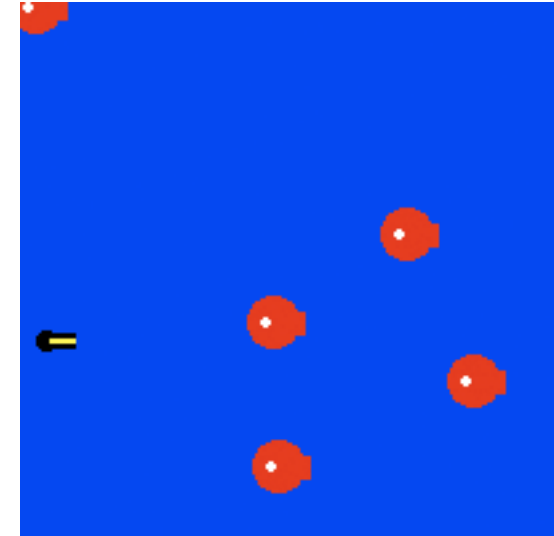
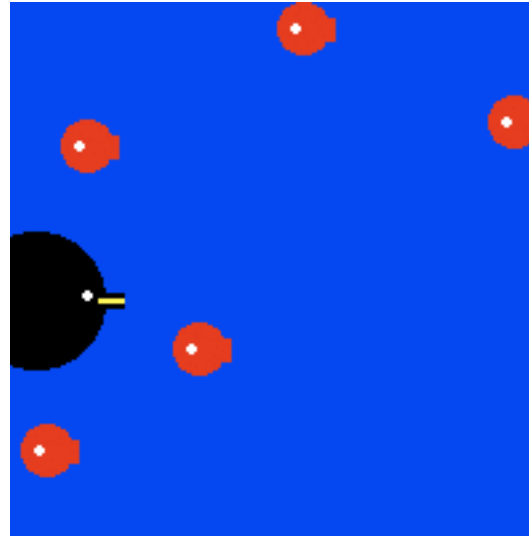
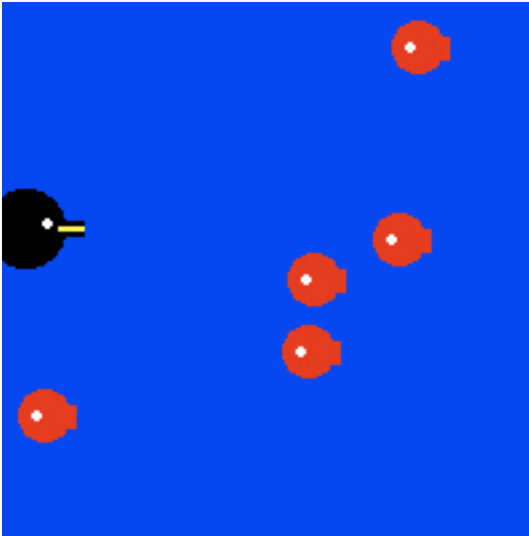
### Language-driven programming

- Advantage: *real world*
- Advantage: real jobs
- Disadvantage: many levels of complexity
- Disadvantage: no explicit design of abstractions
- Disadvantage: learning bad habits early

# Programming and Design



Let's 'play' with the design of a simple game:



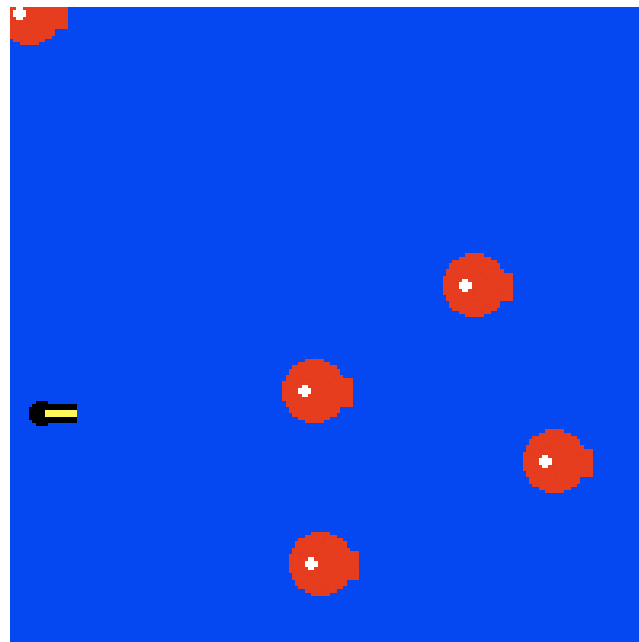
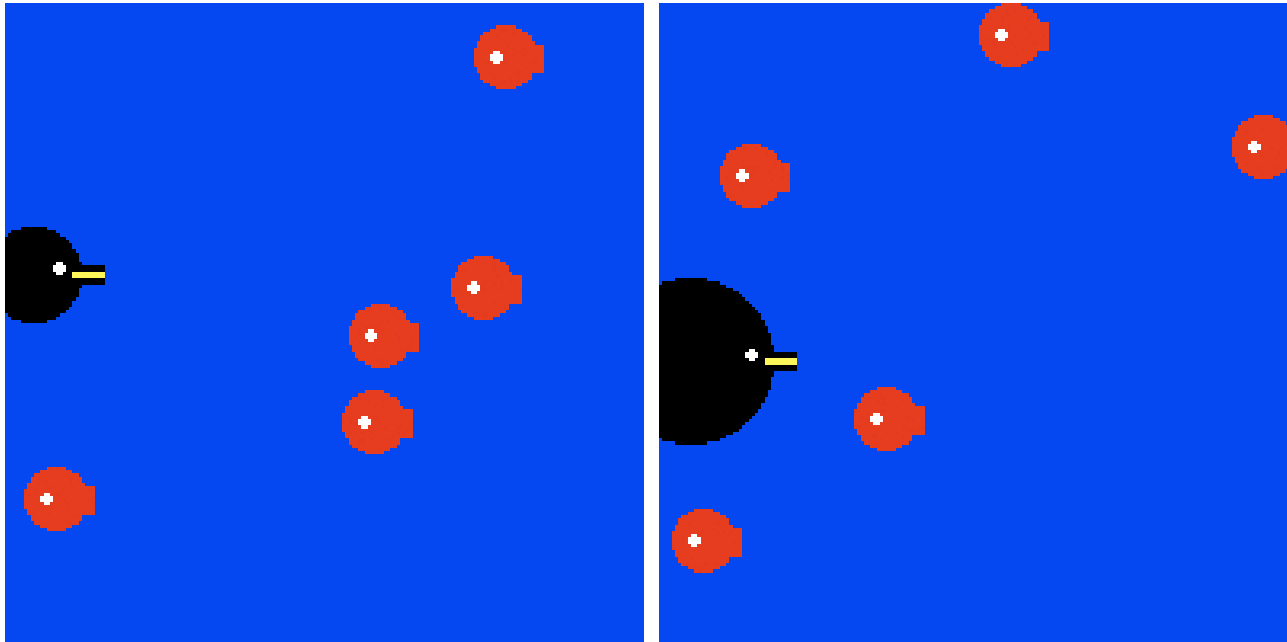
Fish swim across the screen, each is replaced by a new one when it escapes or is eaten

Shark waits, swimming up and down in response to the keys, gets hungrier as the time goes on

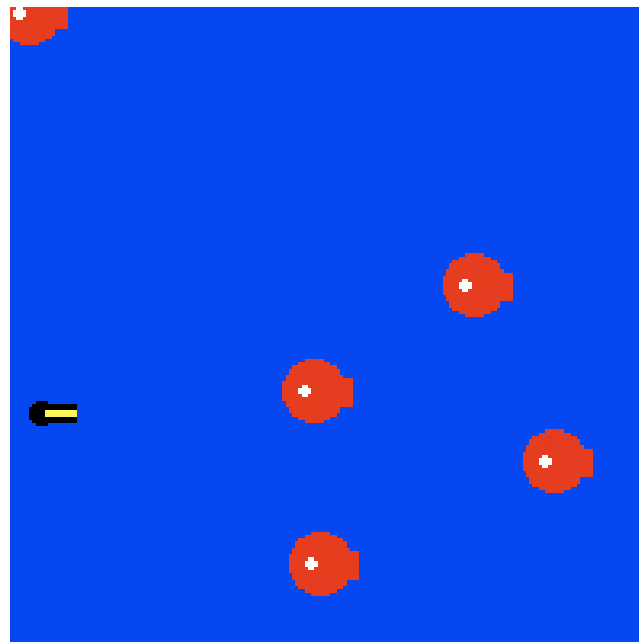
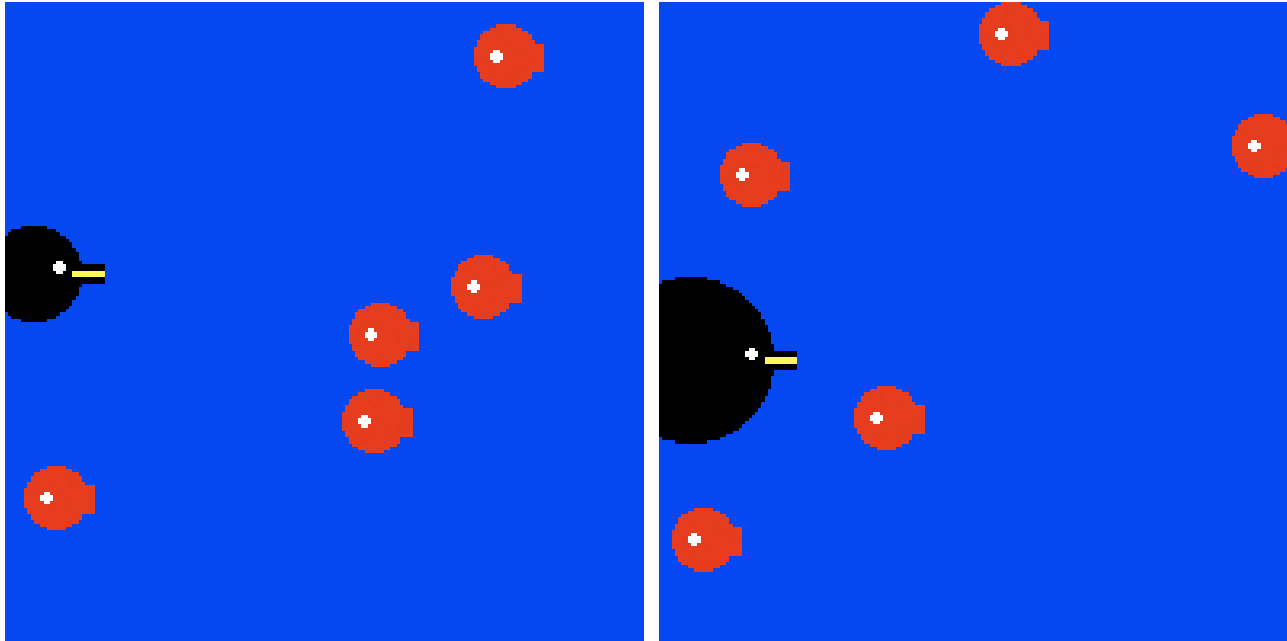
When the shark eats a fish it grows

The game ends when the shark dies of starvation

# Programming and Design



# Programming and Design



# Introduction



## Introductory Computing Options

### Programming as design -- data centric

- **Advantages:** focus on the design - not the language
- **Advantages:** start with simple tasks - systematically
- **Advantages:** introduce language features to support abstractions
- **Advantages:** the student learns to think
- **Disadvantages:** ... this is not the *real world*...???



➤ **What is Introductory Computing?**

➤ **Programming and Design**

- Data vs Information
- Program Design
- The Role of Testing
- Designing Reusable Programs

➤ **Pedagogical Innovations**

➤ **Our Experiences**



# Programming and Design



## The main themes:

- Data vs. Information
- Program Design
- The Role of Testing
- Designing Reusable Programs: Abstractions

## The team:

Matthias Felleisen, Robert Bruce Findler, Matthew Flatt

Kathryn E. Gray, Shriram Krishnamurthi, Viera K. Proulx

# Programming and Design



## Data vs. Information

Think about the problem, what information is available?

- How do we build the game? - what are the parts we need?
  - There is a shark - that moves up and down
  - There is a fish - or more than one - that swims
  - All should stay within the game area

# Programming and Design



## Data vs. Information

Think about the problem, what information is available?

- How do we build the game? - what are the parts we need?
  - There is a **shark** - that moves up and down
  - There is a **fish** - or more than one - that swims
  - All should stay within the **game area**

# Programming and Design



## Data vs. Information

Think about the problem, what information is available?

- **Shark:** what do we know about him?
  - where is the shark
  - how hungry is the shark
- **Fish:** where is the fish?
  - How fast is it swimming?
  - Did it swim out of the game area?
- **Game area:** how wide, how tall?
  - Background color?

# Programming and Design



## Data vs. Information

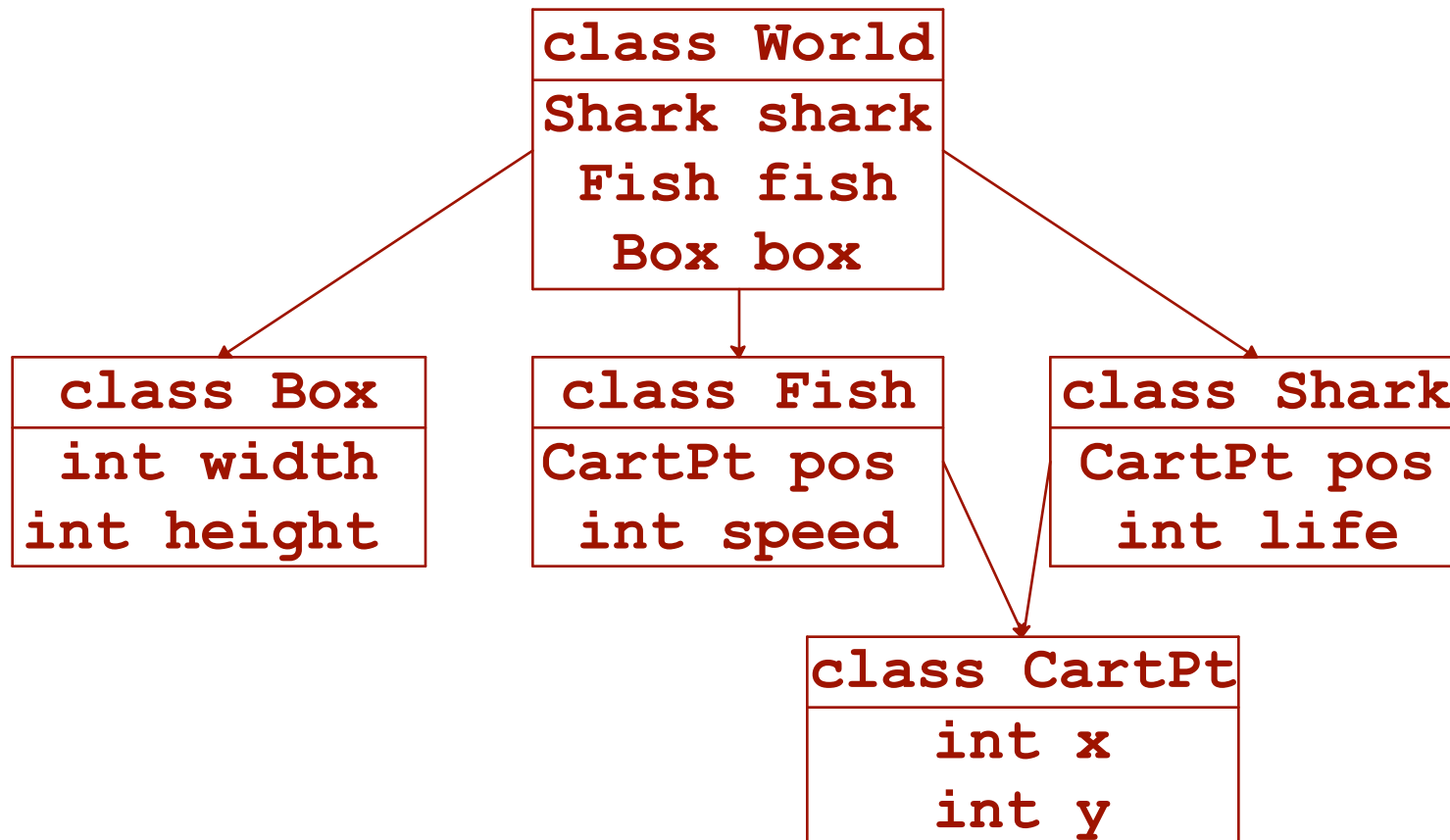
- **World** consists of the area, the fish and the shark
  - Position - consists of the x and y coordinate
  - Life time remaining
- **Shark**
  - Position - consists of the x and y coordinate
  - Life time remaining
- **Fish**
  - Position - consists of the x and y coordinate
  - ... maybe the speed
- **Game area**
  - width and height
  - we also have to draw the shapes

# Programming and Design



## Data vs. Information

Data definition for the world with `CartPt` : a class diagram



# Programming and Design



## Data vs. Information

### Sample data

```
Fish fish = new Fish(new CartPt(200, 100), 5);  
Shark shark = new Shark(new CartPt(20, 100), 30);  
Box box = new Box(200, 200);  
World w = new World(fish, shark, box);
```

# Programming and Design



## Data vs. Information

### Sample data

**Fish fish = new Fish(new CartPt(200, 100), 5);**  
a fish that swims at speed 5 starting from the mid-right of the box

**Shark shark = new Shark(new CartPt(20, 100), 30);**  
a shark with 30 lives starting 20 pixels in from the mid-left of the box

**Box box = new Box(200, 200);**  
the box of width and height 200

**World w = new World(fish, shark, box);**  
the scene 200 by 200 with one fish on the right, one shark on the left



# Programming and Design



## Data vs. Information

- This is complicated enough to warrant separate attention
- We must make sure students understand what data the program works with
- Design Recipe for Data Definition:
  - can it be represented by a primitive type? - select the type
  - are there several parts that represent one entity? - a class
  - are there several related variants? - a union of classes
  - add arrows to connect data definitions
- **Convert information to data**
- **Interpret data as information**

# Programming and Design



## Data vs. Information

### Sample data

**Fish fish = new Fish(new CartPt(200, 100), 5);**  
a fish that swims at speed 5 starting from the mid-right of the box

**Shark shark = new Shark(new CartPt(20, 100), 30);**  
a shark with 30 lives starting 20 pixels in from the mid-left of the box

**Box box = new Box(200, 200);**  
the box of width and height 200

**World w = new World(fish, shark, box);**  
the scene 200 by 200 with one fish on the right, one shark on the left

# Programming and Design



## Designing the functionality

- Move the shark up and down in response to the arrow keys
- Move the fish left as the time goes on
- Replace the fish with a new one if it gets out of bounds
- Check if the shark ate the fish - if yes, replace the fish with a new one
- Starve the shark as the time goes on, check if he is dead

# Programming and Design



## Designing the functionality

- Move the shark up and down in response to the arrow keys
- Move the fish left as the time goes on
- Replace the fish with a new one if it gets out of bounds
- Check if the shark ate the fish - if yes, replace the fish with a new one
- Starve the shark as the time goes on, check if he is dead

# Programming and Design



## Designing the program

- How do you eat an elephant? - one bite at a time
  - One task — one function/method
  - Make a wish list if the task is too complex
  - Think systematically about each small task

# Programming and Design



## Designing the program

- One task — one function or method
- Make a wish list if the task is too complex
- Think systematically about each small task

# Programming and Design



## Select a sub-problem

- Move the shark up and down in response to the arrow keys
- Move the fish left as the time goes on
- Replace the fish with a new one if it gets out of bounds
- Check if the shark ate the fish - if yes, replace the fish with a new one
- Starve the shark as the time goes on, check if he is dead

# Programming and Design



## One Task --- One Function/Method

- Check if the shark ate the fish
- Replace the fish with a new one



# Programming and Design



One Task --- One Function/Method

Check if the shark ate the fish

Replace the fish with a new one

put the second task on a wish list

# Programming and Design



## Designing a Method: Step 1

Check if the shark ate the fish

What data do we need?

-- one **Shark** and one **Fish**

What class is responsible for this task?

-- could be either - choose **Shark**

-- the **Fish** becomes the method argument

What type of result do we produce?

-- a **boolean** value

# Programming and Design



## Designing a Method: Step 2

### Purpose Statement and a Header:

In the class `Fish` :

```
// check if this shark ate the given fish  
boolean ateFish(Fish fishy){...}
```

What should we do next?

... well, when can the shark eat the fish?

... -- when they are close enough to each other

# Programming and Design



## Designing a Method: Step 3

### Examples with Expected Outcomes:

```
// check if this shark ate the given fish  
boolean ateFish(Fish fishy){...}
```

The method produces a **boolean** result

... we need at least two examples

The shark and the fish far away from each other

The shark and the fish are close to each other

# Programming and Design



## Designing a Method: Step 3

### Examples with Expected Outcomes:

```
// check if this shark ate the given fish  
boolean ateFish(Fish fishy){...}
```

```
Fish fish1 = new Fish(new CartPt(200, 100), 5);  
Fish fish2 = new Fish(new CartPt(25, 100), 5);  
Shark shark = new Shark(new CartPt(20, 100), 30);
```

```
shark.ateFist(fish1) ... expect false  
shark.ateFist(fish2) ... expect true
```

# Programming and Design



## Designing a Method: Step 4

What should we do next?

Make an inventory of what we know about the shark and the fish

```
// check if this shark ate the given fish
boolean ateFish(Fish fishy){...}
```

```
this.loc          -- CartPt
this.life         -- int
fishy.loc         -- CartPt
fishy.speed       -- int
```

it depends on how close are the `this.loc` and `fishy.loc`

# Programming and Design



## Designing a Method: Step 4 Inventory/Template

```
// check if this shark ate the given fish
boolean ateFish(Fish fishy){...}
```

```
this.loc          -- CartPt
this.life         -- int
fishy.loc         -- CartPt
fishy.speed      -- int
```

it depends on how close are the `this.loc` and `fishy.loc`

Remember: one task — one function/method

Design a method `boolean distTo(CartPt that)` in the class `CartPt`

# Programming and Design



## Designing a Method: Step 4 Inventory/Template

```
// check if this shark ate the given fish
boolean ateFish(Fish fishy){...}
```

```
this.loc          -- CartPt
this.life         -- int
fishy.loc         -- CartPt
fishy.speed      -- int
```

Design a method in the class CartPt

```
// compute the distance of this point to that
boolean distTo(CartPt that)
```



# Programming and Design



## Designing a Method: Step 4 Inventory/Template

```
// check if this shark ate the given fish
boolean ateFish(Fish fishy){...}
```

```
this.loc           -- CartPt
this.life          -- int
fishy.loc          -- CartPt
fishy.speed        -- int
this.loc.distTo(fishy.loc) -- int
```

Design a method in the class `CartPt`

```
// compute the distance of this point to that
boolean distTo(CartPt that)
```

# Programming and Design



## Designing a Method: Step 5

What should we do next?

We are now ready to design the body of the method

... one question remains:

-- how close does the fish have to be for the shark to eat it?

-- we decide it must be within 20

-- of whatever unit we use to measure the distance

**Here is the complete method - we hope:**

```
// check if this shark ate the given fish
boolean ateFish(Fish fishy){
    return this.loc.distTo(fishy.loc) < 20;}

```

Are we done? ... **NO**

# Programming and Design



## Designing a Method: Step 6

What else needs to be done?

... how do we know we are correct?

... does the method work as we expected it to?

**We already have examples with the expected outcomes!**

Convert the examples into tests and test the method

```
// check if this shark ate the given fish
boolean ateFish(Fish fishy){
    return this.loc.distTo(fishy.loc) < 20;}

```

# Programming and Design



## Designing a Method: Step 6 Tests

```
// check if this shark ate the given fish
boolean ateFish(Fish fishy){
    return this.loc.distTo(fishy.loc) < 20;}

```

```
Fish fish1 = new Fish(new CartPt(200, 100), 5);
Fish fish2 = new Fish(new CartPt(25, 100), 5);
Shark shark = new Shark(new CartPt(20, 100), 30);

```

```
checkExpect(shark.ateFist(fish1), false);
checkExpect(shark.ateFist(fish2), true);

```

# Programming and Design



## Designing a Method: Step 6 Tests

```
// check if this shark ate the given fish
boolean ateFish(Fish fishy){
    return this.loc.distTo(fishy.loc) < 20;}

```

```
Fish fish1 = new Fish(new CartPt(200, 100), 5);
Fish fish2 = new Fish(new CartPt(25, 100), 5);
Shark shark = new Shark(new CartPt(20, 100), 30);

```

```
checkExpect(shark.ateFist(fish1), false);
checkExpect(shark.ateFist(fish2), true);
... add more tests if needed

```

# Programming and Design



## Designing a Method: The DESIGN RECIPE

- 1: Problem analysis and data definition
- 2: Purpose statement and the header
- 3: Examples with expected outcomes
- 4: Inventory/Template of available data fields and methods
- 5: Method body
- 6: Tests

Each step is well defined

-- with a tangible result

-- with a guidance on what questions to ask

# Programming and Design



Other sub-problems --- use the same design process

- Move the shark up and down in response to the arrow keys
- Move the fish left as the time goes on
- Replace the fish with a new one if it gets out of bounds
- Check if the shark ate the fish - if yes, replace the fish with a new one
- Starve the shark as the time goes on, check if he is dead

# Programming and Design



## A complete program:

```
// to represent an ocean world
class OceanWorld extends World{
    Shark shark;
    ILoFish fish;
    int WIDTH = 200;
    int HEIGHT = 200;

    OceanWorld(Shark shark, ILoFish fish) {
        this.shark = shark;
        this.fish = fish;
    }

    // start the world and the timer
    boolean go() { return this.bigBang(200, 200, 0.05); }

    // produce a new OceanWorld after one minute elapsed:
    // move the fish, starve the shark, check if the fish is eaten or has escaped
    World onTick(){
        // if the shark found fish, fed the shark, replace the fish with a new one
        if (this.fish.isFood(this.shark)){
            return new OceanWorld(this.shark.getFatter(),
                this.fish.feedShark(shark));
        }

        // if the shark starved to death, end the world
        else {if(this.shark.isDead()) {
            return this.endOfWorld("The shark starved to death");
        }

        // no special events, just move the fish and starve the shark
        else {
            return new OceanWorld(this.shark.onTick(), this.fish.onTick());
        }
    }
}
```



# Programming and Design



The code for the fish and the shark not shown

-- all completely designed by the student

Student really understands the information and the data

## What makes this possible?

Focus on understanding the data - information first

Testing support

# Programming and Design



## Testing Support

Java does not support comparing data by value

Defining such equality is hard for a novice

It increases the program complexity

Detracts from the focus on the program design

Learning to design tests, equality comparison, test reporting

-- is a topic on its own

-- we need pedagogy for that too

# Programming and Design



## Designing Abstractions

A skill on its own: transcends programming

- motivated by observing repeated code patterns
- students are taught to design abstractions
- each abstraction motivates a new language construct or style

## Java by Demand

# Programming and Design



## Designing Abstractions

Abstractions --- integrated throughout the course

- motivated by observing repeated code patterns
- students are taught to design abstractions

### **Designing abstractions:** Design Recipe for Abstractions

- Identify the differences between similar solutions
- Replace the differences with parameters and rewrite the solution
- Rewrite the original examples and test them again

# Programming and Design



## Designing Abstractions - Motivating Abstractions

### Abstracting over similarities:

- Classes with similar data ➔ abstract classes/interfaces
- Lists of different data ➔ list of  $\langle T \rangle$  ➔ generics
- Classes with similar structure and methods ➔ Abstract Data Types
- Comparisons ➔ interfaces that represent a function object
- Traversal of a container ➔ iterator

# Programming and Design



## Designing Abstractions - Examples of Abstractions

- **Abstract classes:** *common fields, common concrete methods*
- **Generics:** *common structure of data*
  - e.g. *list of  $\langle T \rangle$*
- **Comparable, Comparator:** *common functional behavior*
- **Abstract Data Type**  
*common functional representation of structures*
  - *add, remove, size, contains*
- **Iterators:** *abstracting over traversals*

# Programming and Design



## Designing Abstractions - Why Teach Abstractions?

Eliminate code duplication - reduce maintenance costs

Design reusable code

Build libraries

Learn to use libraries



- **What is Introductory Computing?**
- **Programming and Design**
- **Pedagogical Innovations**
  - Supporting the Novice Programmer: Language Levels
  - Teachpacks: Libraries for Novices
  - Testing Support
  - Self-Regulatory Learning
  - Pedagogical Intervention
- **Our Experiences**



# Pedagogical Innovations



## Programming Environment Support:

- Reduce the syntax/complexity to what is necessary
- Allow the student to focus on the key concepts
- Feedback / error messages at user's level of understanding
- Prevent misuse of advanced features
- Libraries for interactive graphics and games
- Support a well documented test design

Add new features when the need becomes compelling

# Pedagogical Innovations



## Supporting the Novice Programmer: Language Levels

Programming language support at the novice level

- several levels of Java-like languages
- complexity added when student understands more
- new features support new program abstractions
- error messages are appropriate for a novice programmer

# Pedagogical Innovations



## Teachpacks: Libraries for Novices

Libraries that deal with graphics, events

- provide a novice-friendly environment
- hide the interaction with the system
- functional or imperative style
- work the same way in teaching languages and standard Java
- applets in standard Java

# Pedagogical Innovations



## Testing Support

### Test library

Tests are written as a part of the program design

Test library suitable for the beginner

- Tests compare data by their values
  - handle collections of data
  - handle circularity
  - handle random choice
  - handle tests of Exceptions
  - ... and more
- Test evaluation is automatic - compares data by their values

# Pedagogical Innovations



## Self-Regulatory Learning

Theory: encourage the learner to learn on her own

- identify steps in the learning process
- provide a guidance in how to achieve the next step
- provide a way to assess the success of each step

# Pedagogical Innovations



## Self-Regulatory Learning

### Our Practice: The DESIGN RECIPE

- provides the steps in the data, program, abstraction design
- provides questions to ask at each step
- provides a way to assess the success of each step

# Pedagogical Innovations



## Pedagogical Intervention

Instructor asks at which step the student is stuck - then follows with the questions for that step

One more illustration of why and how it works

# Pedagogical Innovations



## Pedagogical Intervention - Self-Regulatory Learning

### Design recipe for designing classes:

#### The problem statement

- we would like to paint geometric shapes -- circles, squares, and combo-shape; see if they overlap and see if a point is inside a shape ...



# Pedagogical Innovations



## Pedagogical Intervention - Self-Regulatory Learning

### Design recipe for designing classes:

#### The problem statement

- we would like to paint geometric shapes -- circles, squares, and combo-shape; see if they overlap and see if a point is inside a shape ...

#### Data Definition- in (key)words

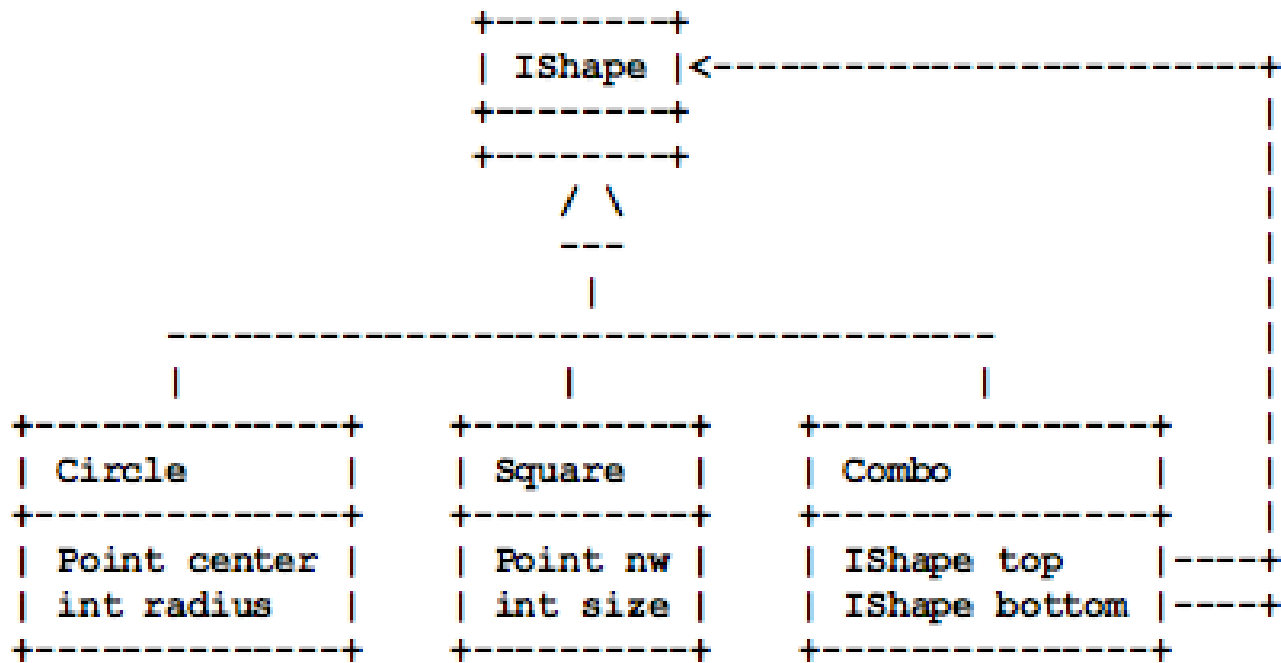
- A Shape is one of:
  - circle: given by a center point and the radius
  - square: given by the NW point and the size
  - combo: given by the top shape and the bottom shape

# Pedagogical Innovations



## Pedagogical Intervention - Self-Regulatory Learning

Class diagram for the IShape class hierarchy:



Corresponds exactly to the narrative data definition

# Pedagogical Innovations



## Pedagogical Intervention - Self-Regulatory Learning

**Design Recipe:** the steps in the design process:

- Problem Analysis and Data Definition -- **understand**
- Purpose & Header -- **interface and documentation**
- Examples -- **show the use in context: design tests**
- Template -- **make the inventory of all available data**
- Body -- **only design the code after tests/examples**
- Test -- **convert the examples from before into tests**

Clear set of questions to answer for each step

Outcomes that can be checked for correctness and completeness

Opportunity for *pedagogical intervention*

# Pedagogical Innovations



## Pedagogical Intervention - Self-Regulatory Learning

**Design Recipe:** the steps in the design process:

- Problem Analysis and Data Definition -- **understand**
- Purpose & Header -- **interface and documentation**
- Examples -- **show the use in context: design tests**
- Template -- **make the inventory of all available data**
- Body -- **only design the code after tests/examples**
- Test -- **convert the examples from before into tests**

**Design foundation:**

- Required documentation from the beginning
- Test-driven design from the beginning
- Focus on the structure of data and the structure of programs



- **What is Introductory Computing?**
- **Programming and Design**
- **Pedagogical Innovations**
- **Our Experiences**
  - University Dissemination
  - Resources

## Our Experiences



Instructors in follow-up courses feel students are much better prepared

Very low attrition rate (<5%)

Students are much more confident in their understanding of program design

### **Dissemination:**

Two very successful summer workshops for secondary school and university teachers in 2003, 2004

Workshop in summer 2007, 2008, 2009 at four US locations

A growing number of followers

# THANK YOU



## Resources:

### Web sites:

Main site for the TeachScheme/ReachJava! project:  
**<http://www.teach-scheme.org>**

Lab materials, lecture notes, assignments:  
**<http://www.ccs.neu.edu/home/vkp/HtDC.html>**

World libraries, Tester library: **<http://www.ccs.neu.edu/javalib>**

Java Power Tools: **<http://www.ccs.neu.edu/jpt>**