Game Design in Object-Oriented Style:

Data, Tests, Programs

Viera Krňanová Proulx

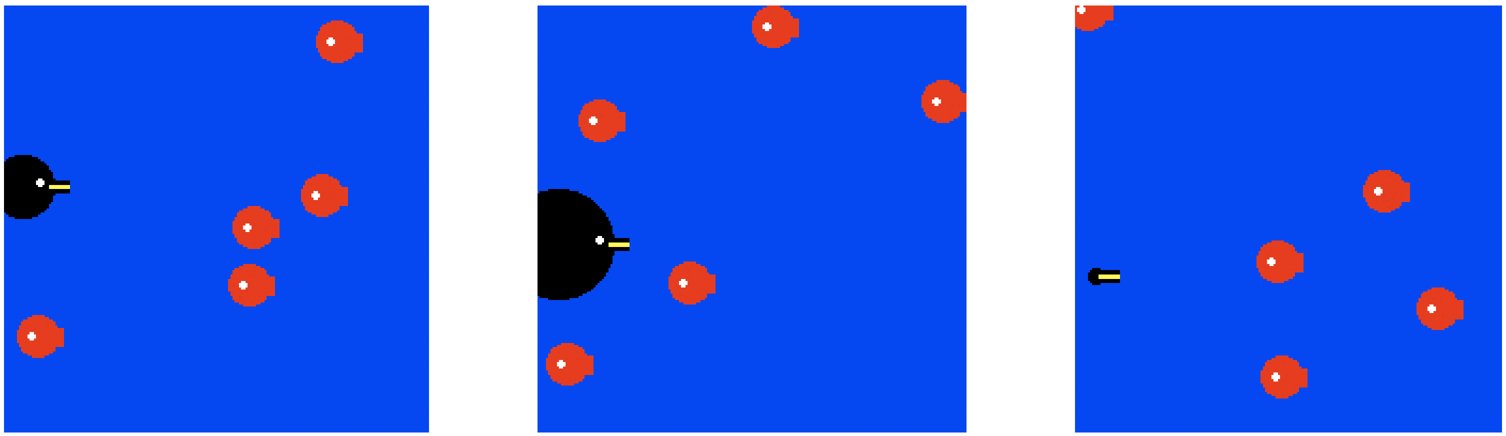
Northeastern University

vkp@ccs.neu.edu

- **Introduction**
  - Overview
- **Programming and Design**
- **Pedagogical Innovations**
- **Our Experiences**

Programming and Design

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



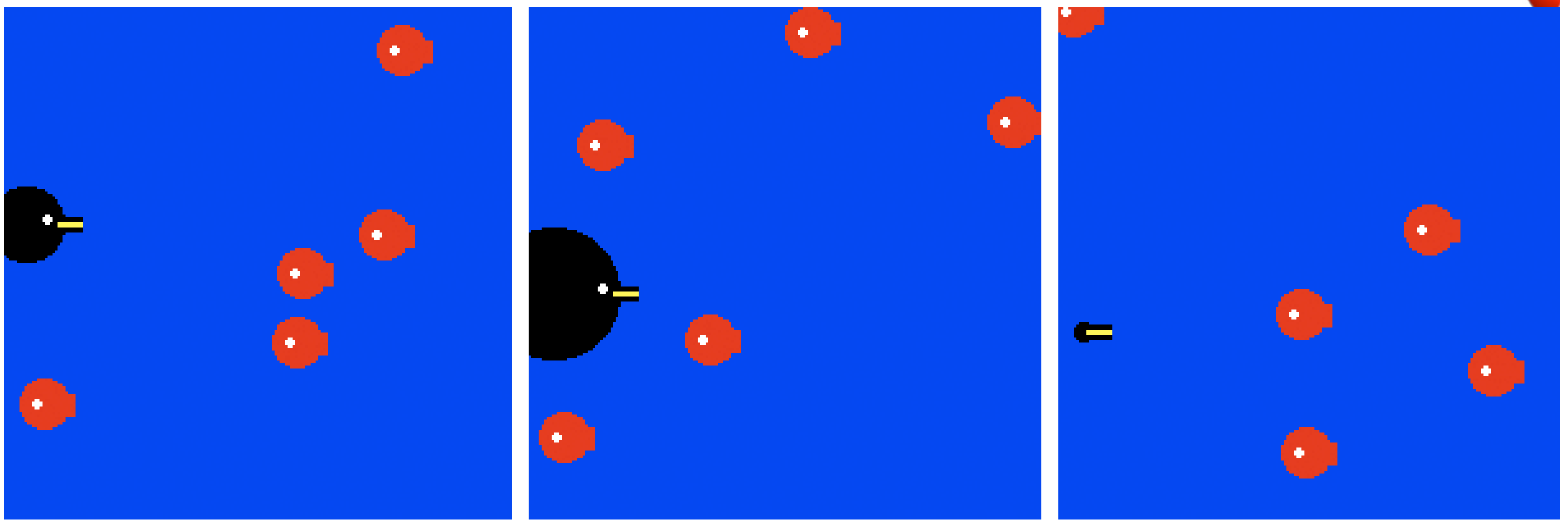
Fish swim across the screen, each is replaced by a new one when it escapes or is eaten

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

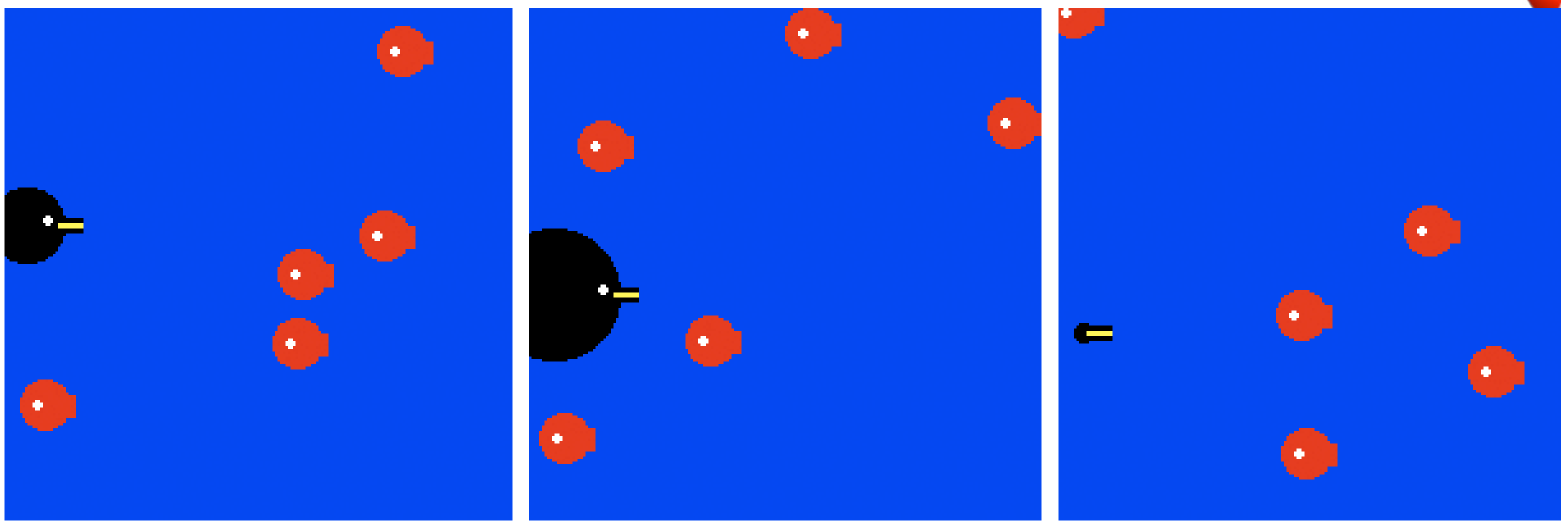


- How do we build the game?
  - We need a frame, a panel with graphics to draw
  - We need to learn how to use the **`Timer`**
  - We need to learn how to use the **`onKeyEventListener`**

Then we can start thinking about the game actions

Technical details hide the program design

Programming and Design



- How do **we** build the game? - what are the parts we need?
  - How do we draw? - make a scene from rectangles, circles with size, color
  - How do we animate? - create a scene for each tick
  - How do we respond to the keys? - define **`onKeyEvent`** method

No complicated system interaction - focus on the game actions

- **Introduction**
- **Programming and Design**
  - Data vs Information
  - Program Design
  - The Role of Testing
  - Designing Reusable Programs
- **Pedagogical Innovations**
- **Our Experiences**

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

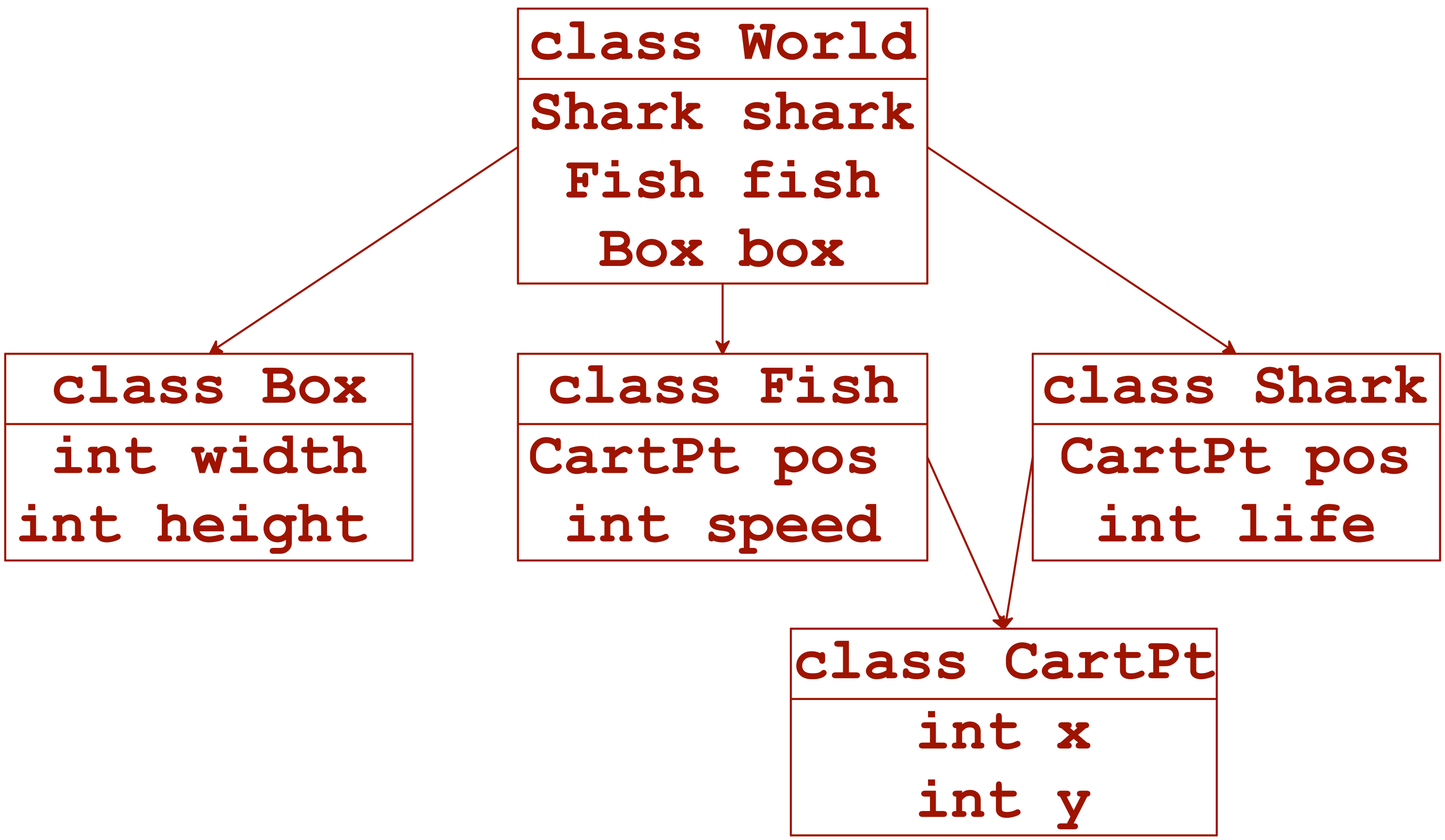
Data vs. Information

- World consists of the area, the fish and the shark
- 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 `Shark`:

```
// 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());
    }}
  }

  // produce a new OceanWorld in response to the given key press
  World onKeyEvent(String ke){
    return new OceanWorld(this.shark.onKeyEvent(ke), this.fish);
  }

  // draw this world
```

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 **<T>** ➡ 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 <T>*
- **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

- **Introduction**
- **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:

```
                    +--------+
                    | IShape |<-------------------------+
                    +--------+                          |
                    +--------+                          |
                       / \                              |
                       ---                              |
                        |                               |
          ------------------------------------          |
          |                 |                 |         |
+---------------+   +-----------+   +----------------+  |
| Circle        |   | Square    |   | Combo          |  |
+---------------+   +-----------+   +----------------+  |
| Point center  |   | Point nw  |   | IShape top     |----+
| int radius    |   | int size  |   | IShape bottom  |----+
+---------------+   +-----------+   +----------------+
```

Corresponds exactly to the narrative data definition

Students use the diagrams to represent the data definition

Method design follows the arrows of the diagram

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

- **Introduction**
- **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**