Back to Basics for APCS Success

Stuart Reges, Principal Lecturer
University of Washington

Hélène Martin, CS teacher
Garfield High School

Selective Timeline

- 1984: AP/CS first offered in Pascal
- 1998-1999: AP/CS switches to C++
- 2001-2002: Dot com crash, CS enrollments plummet
- 2002: OOPSLA “Resolved: Objects have Failed”
- 2003-2004: AP/CS switches to Java
- 2005: SIGCSE “Resolved: Objects Early has Failed”
- 2011: CMU and Berkeley switch CS1 to Python
- 2011: Stuart Reges assures nervous teachers that AP/CS in Java is a fantastic course

More personal timeline

- 2000: “Conservatively Radical Java in CS1”: objects early through scaffolding
- 2004: Stuart hired by UW to fix intro with a plan to teach procedural Java
- 2005: Resolved: Objects Early has Failed
- 2006: first edition of *Building Java Programs*
- 2011: 4 textbooks with “objects late” in title, 3rd edition of *Building Java Programs*

UW Results

Course Principles

- Traditional procedural approach (back to basics): drawing on past wisdom
- Updated to use features of Java: using objects early, graphics (DrawingPanel)
- Core of the course: challenging assignments many of which are nifty or practical
- Concrete practice problems to build programming skills: section problems, labs, exams, PracticeIt
- Lots of support: army of undergraduate TAs, programming lab support

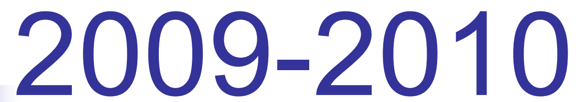
Why I’m sold

- “I've never come across a textbook that layers ideas so **strategically** and ingeniously well. The ideas are presented in an order and in a manner that made it impossible for me to get **lost** or **bored**.

  [...] It taught so well, I couldn't wait to get my hands on problem after problem. This book made me **crave problem solving** and writing clean, inventive, non-redundant, well-commented code.”

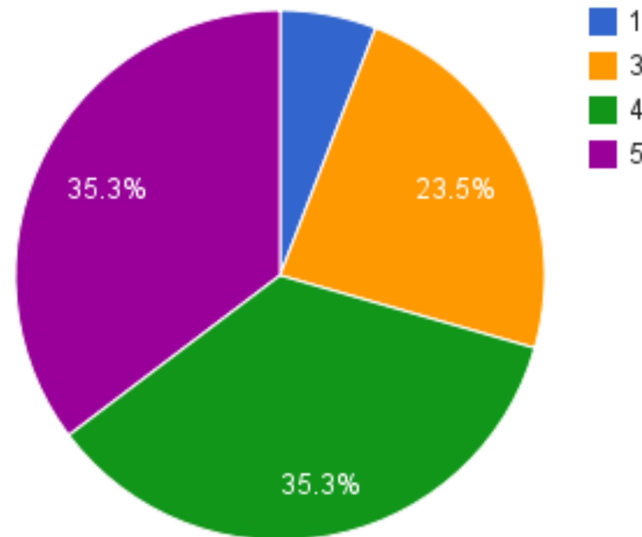
  - \- Amazon review

- Applies to methodology; book is a nice-to-have!



2009-2010

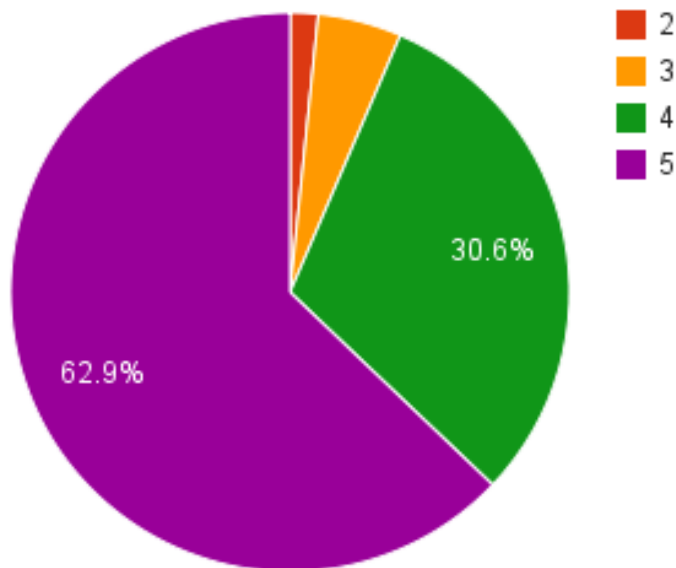
- First offering of APCS in the district
- 26 students enrolled, 17 took AP test

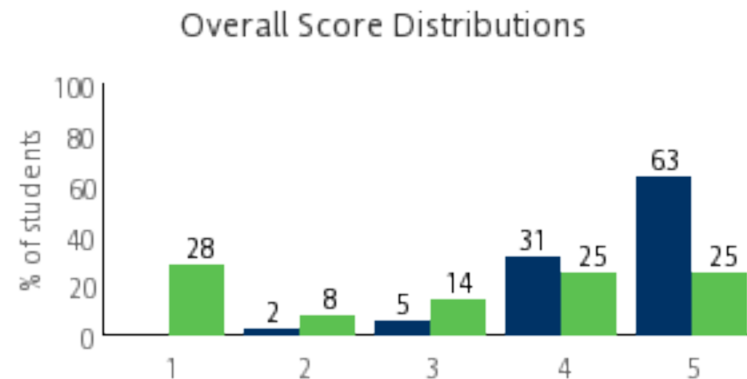


2010-2011

- Advanced section for 25 students
- 2 sections for students new to programming
- 32% women overall (37% in new sections)

Garfield Computer Science 2011 AP Scores (62 students)





Garfield course structure

- 1/4 lecture, group work without computers

- In-class time for experimenting
- Programming projects written from scratch
- Little to no homework

- Bi-weekly paper and pencil quizzes
- No real mention of AP test until February

Students know OOP

- January: writing classes as object blueprints
- Sophisticated Gridworld projects
  - 15-puzzle
  - snake game
  - ant farm
- Heavily OO final projects
- AP report mean for OO multiple choice: 6.4, 4.9 nationally; group mean close to 7 on FRQ

Assertions: verifying mental models

```
public static void mystery(int x, int y) {
    int z = 0;
    // Point A

    while (x >= y) {
        // Point B
        x = x - y;
        z++;

        if (x != y) {
            // Point C
            z = z * 2;
        }
        // Point D

    }
    // Point E
    System.out.println(z);
}
```

Which of the following assertions are true at which point(s) in the code? Choose ALWAYS, NEVER, or SOMETIMES.

	`x < y`	`x == y`	`z == 0`
Point A			
Point B			
Point C			
Point D			
Point E			

Assertions: verifying mental models

```
public static void mystery(int x, int y) {
    int z = 0;
    // Point A

    while (x >= y) {
        // Point B
        x = x - y;
        z++;

        if (x != y) {
            // Point C
            z = z * 2;
        }
        // Point D

    }
    // Point E
    System.out.println(z);
}
```

Which of the following assertions are true at which point(s) in the code? Choose ALWAYS, NEVER, or SOMETIMES.

	x < y	x == y	z == 0
Point A	SOMETIMES	SOMETIMES	ALWAYS
Point B	NEVER	SOMETIMES	SOMETIMES
Point C	SOMETIMES	NEVER	NEVER
Point D	SOMETIMES	SOMETIMES	NEVER
Point E	ALWAYS	NEVER	SOMETIMES

Reasoning about assertions

- Right after a variable is initialized, its value is known:

```
int x = 3;
// is x > 0?  ALWAYS
```

- In general you know nothing about parameters' values:

```
public static void mystery(int a, int b) {
// is a == 10?  SOMETIMES
```

- But inside an `if`, `while`, etc., you may know something:

```
public static void mystery(int a, int b) {
    if (a < 0) {
        // is a == 10?  NEVER
        ...
    }
}
```

Assertions and loops

- At the start of a loop's body, the loop's test must be `true`:

```
while (y < 10) {
    // is y < 10?  ALWAYS
    ...
}
```

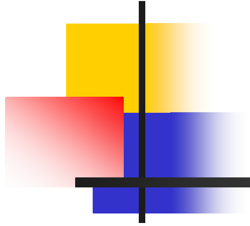
- After a loop, the loop's test must be `false`:

```
while (y < 10) {
    ...
}
// is y < 10?  NEVER
```

- Inside a loop's body, the loop's test may become `false`:

```
while (y < 10) {
    y++;
    // is y < 10?  SOMETIMES
}
```

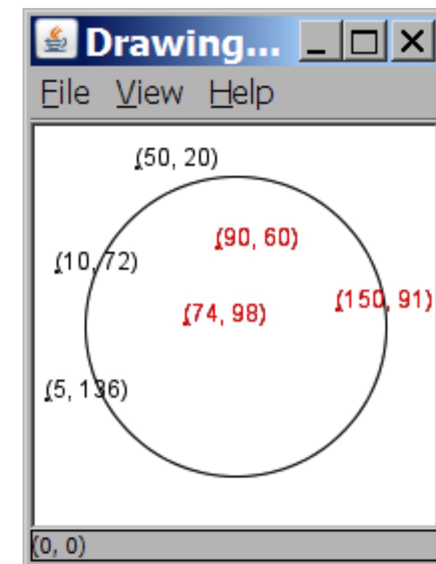
- Things that cause a variable's value to be unknown:
  - reading from a `Scanner`
  - choosing a random value
  - a parameter's initial value to a method



Transition to OOP

Modeling earthquakes

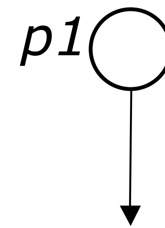
- Given a file of cities' (x, y) coordinates, which begins with the number of cities:
  - `6`
  - `50 20`
  - `90 60`
  - `10 72`
  - `74 98`
  - `5 136`
  - `150 91`



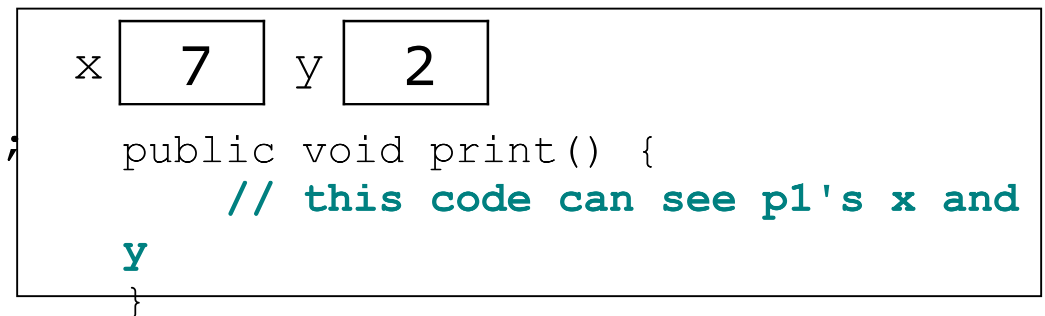
- Write a program to draw the cities on a `DrawingPanel`, then model an earthquake by turning affected cities red:
  - `Epicenter x?` **100**
  - `Epicenter y?` **100**
  - `Affected radius?` **75**

`Point` objects w/ method

- Each `Point` object has its own copy of the `print` method, which operates on that object's state:

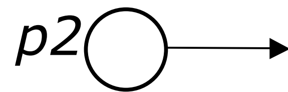


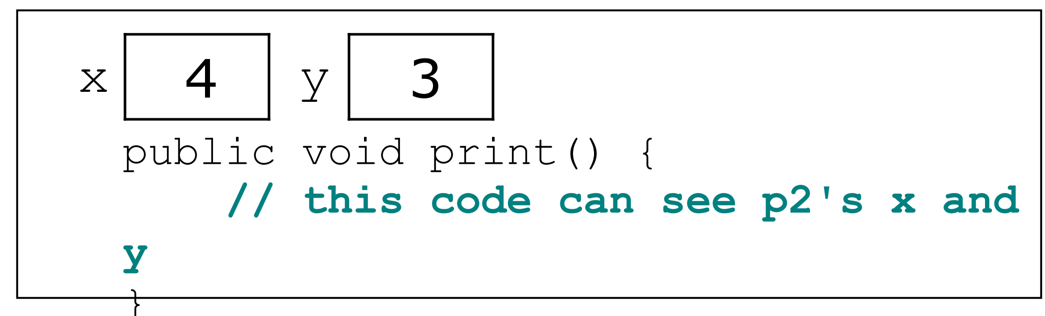
  - `Point p1 = new Point();`
  - `p1.x = 7;`
  - `p1.y = 2;`



  - `Point p2 = new Point();`
  - `p2.x = 4;`
  - `p2.y = 3;`

  - **`p1.print();`**
  - **`p2.print();`**





Why encapsulation?

- Abstraction between object and clients
- Protects object from unwanted access
  - Example: Can't fraudulently increase an `Account`'s balance.
- Can change the class implementation later
  - Example: `Point` could be rewritten in polar coordinates (r, θ) with the same methods.
- Can constrain objects' state (**invariants**)
  - Example: Only allow `Account`s with non-negative balance.
  - Example: Only allow `Date`s with a month from 1-12.