CS 161: Object Oriented Problem Solving

About this course

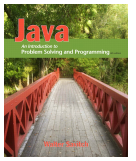
- Like 160, 161 is a combination of programming and discrete math.
  - Why is math important to us? What does that have to do with computer science?
- From procedural to object oriented programming

About this course

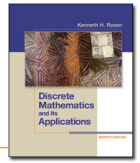
- Course webpage: http://www.cs.colostate.edu/~cs161/
  - The course webpage is our major communication tool. Check it on a daily basis!
  - RamCT will be used for: ~~forums~~, grades
  - Piazza instead of RamCT for forums

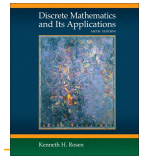
Texts

- Java: an introduction to problem solving and programming, 5th or 6th edition.



- Discrete Mathematics and Its Applications, 6th or 7th edition. Kenneth Rosen.





Components of the course

- Quizzes & class participation:
  "are you with us?" Not worth many points but useful finger on the pulse (for you and me)
- Tests:
  "what have you learned?" Important checkpoints!!
- Programming assignments:
  "can you implement it?" Important!! You get a chance to learn from your mistakes (regrade)
- Written assignments:
  "do you understand the theory?"

About this course

- Lectures
  - You pay for them, might as well use them
  - We are here to help
- Recitation
  - Help you with programming and homework assignments, reinforce material from lecture.
  - you get credit for attending and participating in recit

Grading

Assignments
Quizzes and participation
Recitation (attendance + completion)
Midterms (2) one of these is a programming midterm
Final

For the percentage break down see web page. You need to have a passing grade on the exams (>= 60) to get a passing grade in the course.

CS building

- Make sure you can get into the Unix lab!
  If you have keycard access problems:
  - CS students: talk to any CS front desk person (Kim, Sharon or student employees)

Professional class behavior

- We all have to have respect for each other, independent of race, gender, ability
- THERE ARE NO STUPID QUESTIONS
  - Your classmates will be grateful you asked.

Cheating

- What is cheating? What is not?
  - **Where would you find a definition?**
- What is gained / lost when cheating?
- What are the consequences?
- When / how does it happen?
  - **How can cheating be avoided?**

First topic: Recap of CS160.

- Lecture: Overview of Java program structure
- Recitation: Write simple programs, homework submission via checkin
- Homework assignment 1: More recap

Reading values from a file

Issues with the code?

```
import java.io.*;
import java.util.*;
class FileReader1{
  public static void main(String[ ] args) throws IOException{
      String filename = "values.dat";
      Scanner in = new Scanner(new File(filename));
      int index =  0;
      int[ ] list = new int [9999];
      while(in.hasNext( )){
         list[index] = in.nextInt( );
         index++;
      }
      in.close( );
      // do something with the array
  }
}
```

Methods

```
public class C2F {
    public static void main(String[ ] args){
        C2F converter = new C2F();
        System.out.print("100C eq."); argument
        System.out.print(converter.c2f(100) + " F");
  }
                                parameter
  public double c2f(double celsius) {
      return celsius * 9 / 5 + 32;
  }
}
```

Methods are discussed in section 5.1 in Savitch

Primitive variables

- When a primitive variable is assigned to another, the value is copied.
- Therefore: modifying the value of one does not affect the copy.
- Example:

```
int x = 5;
int y = x;      // x = 5, y = 5
y = 17;         // x = 5, y = 17
x = 8;          // x = 8, y = 17
```

Primitive variables as method arguments

When primitive variables (`int`, `double`) are passed as arguments, their values are copied. Modifying the parameter inside the method will not affect the variable passed in.

```
public void caller() {
    int x = 23;
    strange(x);
    System.out.println("2. Value of x in caller = " + x);
}
public void strange(int x) {
    x = x + 1;
    System.out.println("1. Value of x in strange  = " + x);
}
```

Output:

```
1. Value of x in strange = 24
2. Value of x in caller = 23
```

Reading values from a file (again)

```
import java.io.IOException;
import java.util.Scanner;
class FileReader1{
  public static void main(String[ ] args) throws IOException{
      String filename = "values.dat";
      Scanner in = new Scanner(new File(filename));
      int[ ] list = new int [9999];
      int index =  0;
      while(in.hasNext( )){
         list[index] = in.nextInt( );
         index++;
      }
      in.close( );
      // do something with the array
  }
}
```

Let's improve this version using methods!

```
import java.io.IOException;
import java.util.Scanner;
import java.util.Arrays;
class FileReader2{
   public static void main(String[ ] args) throws IOException {
      String filename = "values.dat";
      ReadFile2 fileReader = new FileReader2();
      int [ ] values = fileReader.readFile(filename);
      System.out.println(Arrays.toString(values));
   }
   public int [ ] readFile(String filename) throws IOException {
      Scanner in = new Scanner(new File(filename));
      int index =  0;
      int[ ] list = new int [9999];
      while(in.hasNext( ))     {
         list[index] = in.nextInt( );
         index++;
      }
      in.close( );
      return list;
   }
}
```

```
import java.io.*;
import java.util.*;
class FileReader3 {
   public static void main(String[ ] args) throws IOException{
      String filename = "values.dat";
      FileReader3 fileReader = new FileReader3();
      int[ ] list = new int [9999];
      fileReader.readFile(filename, list);
      System.out.println(Arrays.toString(list));
   }
   public void readFile(String filename, int [ ] values) throws
      IOException {
      Scanner in = new Scanner(new File(filename));
      int index =  0;
      while(in.hasNext( ))     {
         values[index] = in.nextInt( );
         index++;
      }
      in.close( );
   }
}
```

Objects and the **new** operator

- Compare

  `int[ ] list = new int [9999];`

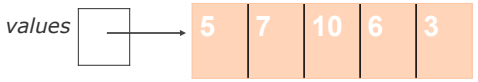
  with

  `int index =  0;`
- Why does one use the new operator while the other doesn't?

Objects and references

- Object variables store the address of the object value in the computer's memory.
- Example:

  `int [] values  = new int[5];`



`int x = 1;`

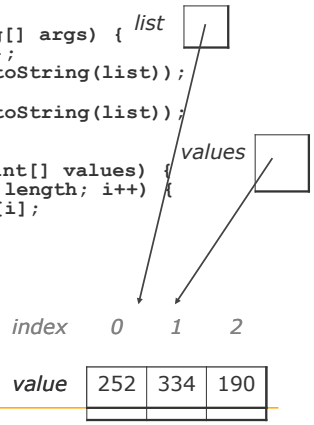
Example

```
public static void main(String[] args) {
    int[] list = {126, 167, 95};
    System.out.println(Arrays.toString(list));
    doubleAll(list);
    System.out.println(Arrays.toString(list));
}

public static void doubleAll(int[] values) {
    for (int i = 0; i < values.length; i++) {
        values[i] = 2 * values[i];
    }
}
```

- **Output:**

```
[126, 167, 95]
[252, 334, 190]
```



Variable scope

- **scope**: The part of a program where a variable exists.
  - A variable's scope is from its declaration to the end of the { } braces in which it was declared.
  - If a variable is declared in a `for` loop, it exists only in that loop.
  - If a variable is declared in a method, it exists only in that method.

```
public static void example() {
    int x = 3;
    for (int i = 1; i <= 10; i++) {
        System.out.println("i=" + i + "x=" + x);
    }
    // i no longer exists here
} // x ceases to exist here
```

Variable scope

- It is illegal to try to use a variable outside of its scope.

```
public static void example() {
    int x = 3;
    for (int i = 1; i <= 10; i++) {
        System.out.println(x);
    }
    System.out.println(i);  // illegal
}
```

Variable scope

- It is legal to declare variables with the same name, as long as their scopes do not overlap:

```
public static void main(String[] args) {
    int x = 2;
    for (int i = 1; i <= 5; i++) {
        int y = 5;
        System.out.println(y);
    }
    for (int i = 3; i <= 5; i++) {
        int y = 2;
        int x = 4;  // illegal
        System.out.println(y);
    }
}
public static void anotherMethod() {
    int i = 6;
    int y = 3;
    System.out.println(i + ", " + y);
}
```