CS 570: Data Structures

Intro to Java

Instructor: Iraklis Tsekourakis

Email: itsekour@stevens.edu

About me

- Prof. Iraklis Tsekourakis
  - Grew up in Kavala, Greece
  - Started programming in BASIC/MS-DOS in my last year of elementary school (1995)
  - Moved to the U.S. 7 years ago for my graduate studies

About me

- Education
  - BS in ECE – Aristotle university of Thessaloniki (2010)
  - MS in CS – Stevens Institute of Technology (2014)
  - PhD in CS – Stevens Institute of Technology (2016)
- Professional Experience
  - Assistant Teaching Professor (2016-present)
  - Research Associate in CERTH/ITI (Information Technologies Institute) (2011-2012)
    - Software development, Project management, Research

Teaching

- I’ve been an instructor on
  - Intro to Web Programming
  - Operating Systems
  - Algorithms
  - Data Structures

Research

- Autonomous Software Agents
- **3-D Computer Vision**
- Machine Learning

Objectives

- Review of Java language.
- Understand the notion of Abstract Data Types, and their use in object-oriented designs.
- Calculate the Big-O of diverse non-recursive algorithms and use it to compare efficiency.
- Use and understand the Collection class in Java, with major emphasis on Lists, Stacks and Queues.
- Implement Binary Search Trees, Max/Min Heaps, Priority Queues in Java, and understand the basic concepts of self-balancing Binary Search Trees.
- Understand what are Sets and Maps, and more specifically implement hash tables in Java.
- Understand and implement recursive algorithms, and data structures.
- Combine different classes together to implement big programming assignments in Java, including a final project that combines some of the data structures studied in class

Important Points

- At any point, ask me WHY?
- USE the virtual office hours
  - If you think a homework is taking too long or is wrong
  - Etc. etc.
- You can ask me anything about the course in my virtual office hours, or by email
- This is a graduate course.
- Expect to work, and to be challenged

Logistics

- Class webpage: CANVAS
  - https://www.stevens.edu/canvas
- Virtual Office hours: Thursday 6-7pm
  - https://sit.instructure.com/courses/34684/external_tools/86566

Logistics II

- Course Evaluation
  - Quizzes 10%
  - Assignments 60%
  - Final 30%

Assignments

- Individual
- Will be done in Java (more details on that later)
- This is a Java Course
  - Programming is arguably the number one skill of a CS graduate
- But I care equally (or slightly more) for algorithms, and data structures than syntax

Resources

- Textbook
  - Data Structures: Abstraction and Design Using Java, 3rd Edition
- Slides and Videos
  - On Canvas

Syllabus

	Topic(s)	Reading(s)	HW
Week 1	Java Syntax	Appendix A	
Week 2	Java Syntax (II), ADT	Appendix A, Chapter 1	**Assignment on Java**
Week 3	Complexity	Chapter 2	
Week 4	Array based Lists, Linked Lists	Chapter 2	**Assignment on Complexity**
Week 5	Double Linked Lists, Iterators, Collections interface	Chapter 2	
Week 6	Stacks	Chapter 3	**Assignment on Double Linked Lists**
Week 7	Queues	Chapter 4	
Week 8	Recursion	Chapter 5	
Week 9	More Recursion	Chapter 5	**Assignment on Recursion**
Week 10	Introduction to Trees	Chapter 6	
Week 11	Binary Search Trees, max/min heaps, priority queues	Chapter 6	**Assignment on Priority Queues**
Week 12	Sets & Maps, Hashing	Chapter 7	
Week 13	Sets & Maps, Hashing	Chapter 7	**Final Programming Project**
Week 14	Sorting	Chapter 8	

Grading Policies

- Late submission?
- Quizzes
- Practice coding exercises
- Individual Assignments: NO collaboration
  - Any sign of collaboration -> Honor Board
- Midterm surveys and not only: Provide feedback!

Week 1 Objective

- Intro to Java
- Reading Assignment: Koffman and Wolfgang Appendix A

- **TODO: Install Eclipse**
- Also requires the Java Development Kit

Java

- WORA: Write Once, Run Anywhere
- Java Virtual Machine
- Other advantages include security, extensibility and low software production cost

Java is Object Oriented

- Classes and Objects
  - Class definitions in .java files
  - Def: a *class* is a named description for a group of entities that have the same characteristics
  - *Objects* or *instances* of the class is the group of entities
  - The characteristics are the attributes (data fields) for each object and the operations (methods) that can be performed on these objects

Data Fields and Types

- Data fields are variables
- Java is strongly typed
  - Every variable must be declared with a data type before it can be used
  - There are built-in types and user-defined types

Primitive Data Types

- byte -128 to 127
- short -32,768 to 32,767
- int -2,147,483,648 to 2,147,483,647
- long -9,223,372,036,854,775,808 to …
- float $\pm 10^{38}$ incl. 0 with 6 digits of precision
- double $\pm 10^{308}$ incl. 0 with 15 digits of precision
- char Unicode character set
- boolean true, false

Methods

- Method: a group of statements to perform a particular operation (called function in many other languages)
- Instance Methods: Applied to an object using dot notation
- `object.method(arguments)`
- E.g. the `println` method that can be applied to `PrintStream` object `System.out`
  - `System.out.println("The value of x is "+x);`

Static Methods

- `static char minChar(char ch1, char ch2){`
- **static** indicates that it is a *static or class method*
  - There is one per class, not one per object like instance methods
- Called using dot notation
  - `char ch=ClassName.minChar('a','A');`
- Static methods cannot call instance methods

Static vs. Instance Methods

```
public class Car{
…
?? float km2Miles(float km)
?? float getOdometerMiles()

}
```

The `main` Method

- Point where execution begins

```
public static void main(
  String[] args){
```

public:

static:

void:

Defining your own classes

- A Java program is a collection of classes
- Let’s see some examples

Rectangle Example

```
public class Rectangle{
	public double width;
	public double height;

	// constructor
	public Rectangle(double x, double y){
		width = x;
		height = y;
	}

	public double area(){
		return width*height;
	}
}
```

Rectangle Example

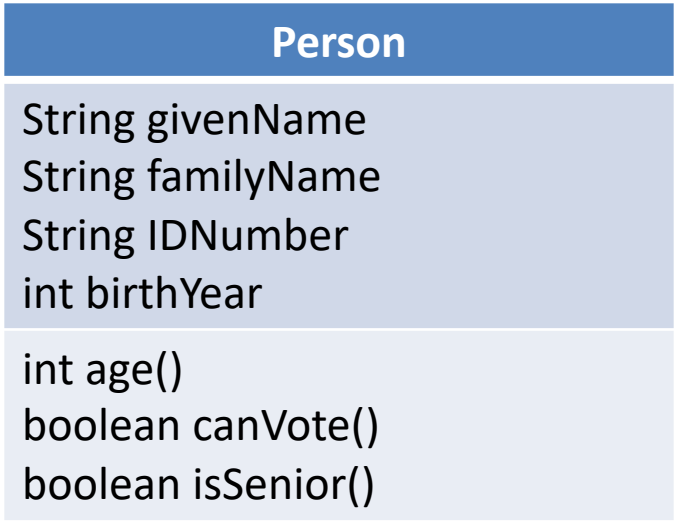
```
// int main() method
// create a rectangle with width 3.5 and height 2.6
Rectangle rect = new Rectangle(3.5, 2.6);

// get its area
double ar;
ar = rect.area();
```

Person Example

- For example:
  - A class Person may store:
    - Given name
    - Family name
    - ID number
    - Year of birth
  - It can perform operations such as:
    - Calculate person's age
    - Test whether two Person objects refer to same person
    - Determine if the person is old enough to vote
    - Get one or more of the data fields from the Person object
    - Set one or more of the data fields of the Person object

UML Diagram



Data fields (instance variables)

Methods

```
/** Person is a class that represents a human being.
 *  @author Koffman and Wolfgang
 * */

public class Person {
  // Data Fields
  /** The given name */
  private String givenName;

  /** The family name */
  private String familyName;

  /** The ID number */
  private String IDNumber;

  /** The birth year */
  private int birthYear = 1900;

  // Constants
  /** The age at which a person can vote */
  private static final int VOTE_AGE = 18;

  /** The age at which a person is considered a senior citizen */
  private static final int SENIOR_AGE = 65;
```

```
// Constructors
/** Construct a person with given values
    @param first The given name
    @param family The family name
    @param ID The ID number
    @param birth The birth year
 */
public Person(String first, String family, String ID, int birth) {
  givenName = first;
  familyName = family;
  IDNumber = ID;
  birthYear = birth;
}

/** Construct a person with only an IDNumber specified.
    @param ID The ID number
 */
public Person(String ID) {
  IDNumber = ID;
}
```

```
// Modifier Methods
/** Sets the givenName field.
    @param given The given name
 */
public void setGivenName(String given) {
  givenName = given;
}

/** Sets the familyName field.
    @param family The family name
 */
public void setFamilyName(String family) {
  familyName = family;
}

/** Sets the birthYear field.
    @param birthYear The year of birth
 */
public void setBirthYear(int birthYear) {
  this.birthYear = birthYear;
}
```

```
// Accessor Methods
/** Gets the person's given name.
    @return the given name as a String
 */
public String getGivenName() {
  return givenName;
}

/** Gets the person's family name.
    @return the family name as a
 String
 */
public String getFamilyName() {
  return familyName;
}

/** Gets the person's ID number.
    @return the ID number as a String
 */
public String getIDNumber() {
  return IDNumber;
}

/** Gets the person's year of birth.
    @return the year of birth as an
int value
 */
public int getBirthYear() {
  return birthYear;
}
```

```
// Other Methods
/**  Calculates a person's age at this year's birthday.
     @param year The current year
     @return the year minus the birth year
 */
public int age(int year) {
  return year - birthYear;
}

/** Determines whether a person can vote.
    @param year The current year
    @return true if the person's age is greater than or
            equal to the voting age
 */
public boolean canVote(int year) {
  int theAge = age(year);
  return theAge >= VOTE_AGE;
}
```

```
/** Determines whether a person is a senior citizen.
    @param year the current year
    @return true if person's age is greater than or
            equal to the age at which a person is
            considered to be a senior citizen
 */
public boolean isSenior(int year) {
  return age(year) >= SENIOR_AGE;
}

/** Retrieves the information in a Person object.
    @return the object state as a string
 */
public String toString() {
  return "Given name: " + givenName + "\n"
      + "Family name: " + familyName + "\n"
      + "ID number: " + IDNumber + "\n"
      + "Year of birth: " + birthYear + "\n";
}
```

```
  /** Compares two Person objects for equality.
      @param per The second Person object
      @return true if the Person objects have same
              ID number; false if they don't
   */
  public boolean equals(Person per) {
    if (per == null)
      return false;
    else
      return IDNumber.equals(per.IDNumber);
  }
}
```

Private Data Fields, Public Methods

- Better control of how data is accessed
- Details of how data are stored and represented can be changed without affecting class's clients

Constructors

- Four-parameter
- One-parameter

- No-parameter constructor is not defined
- `Person p = new Person()` is invalid
- No-parameter constructor has to be explicitly defined if other constructors are defined

Use of `this.`

```
public void setBirthYear(int birthYear) {
    this.birthYear = birthYear;
  }
```

- `birthYear` is interpreted by the Java compiler as the local variable (parameter here) and not the data field with the same name

The Method `toString`

- To display the state of author1 (an instance of Person), we could use:
  - `System.out.println(author1.toString());`
  - `System.out.println(author1);`
- `System.out.println` and `System.out.print` automatically apply method `toString()` to an object that appears in their argument list

The Method `equals`

```
public boolean equals(Person per) {
    if (per == null)
      return false;
    else
      return IDNumber.equals(per.IDNumber);
  }
```

- We can look at `per`'s private ID number because `per` references an object of this class (Person)

testPerson

```
public class TestPerson {
  public static void main(String[] args) {
    Person p1 = new Person("Sam", "Jones", "1234", 1930);
    Person p2 = new Person("Jane", "Jones", "5678", 1990);
    System.out.println("Age of " + p1.getGivenName() +
                       " is " + p1.age(2012));
    if (p1.isSenior(2004))
      System.out.println(p1.getGivenName() +
                         " can ride the subway for free");
    else
      System.out.println(p1.getGivenName() +
                         " must pay to ride the subway");

    System.out.println("Age of " + p2.getGivenName() +
                       " is " + p2.age(2012));
    if (p2.canVote(2004))
      System.out.println(p2.getGivenName() + " can vote");
    else
      System.out.println(p2.getGivenName() + " can't vote");
  }
}
```

Arrays

```
int[] scores = new int[5];
String[] names = {“Sally”, “Jill”, “Hal”,
  “Rick”};

Person[] people;
// define n in some way
int n = …
people = new Person[n];
people[0] = new Person(“Elliot”,
  “Koffman”,”123”,1942);
```

`length` is a data field not a method

Arrays of Arrays

```
double[][] matrix = new double[5][10];
```

- In Java, you can have two-dimensional arrays with rows of different sizes

```
char[][] letters = new char [5][];
letters[0] = new char[4];
letters[1] = new char[10];
```

Style

- Camel notation
  - `myVariable, thisLongIdentifier`
- Primitive type constants
  - all caps: `static final int MAX_SCORE=999`
- Postfix/prefix increment
  - `z=i++;`
  - `z=++i;`
  - Don't use `x*++i`

Pre-increment VS post-increment

- `++i` will increment the value of `i`, and then return the incremented value.

```
i = 1;
j = ++i;
(i is 2, j is 2)
```

- `i++` will increment the value of `i`, but return the original value that `i` held before being incremented.

```
i = 1;
j = i++;
(i is 2, j is 1)
```

Type Compatibility and Conversion

- When mixed type operands are used, the type with the smaller range is converted to the type of the larger range
  - E.g. int+double is converted to double
  - *Widening* conversion
  -
```
int item = …;
double realItem = item; // valid ?
```
  -
```
double y = …;
int x=y; // valid ?
```

Referencing Objects

- `String greeting;`
- `greeting = "hello";`
  - String object "hello" is now referenced by `greeting`
  - `greeting` stores the address of where a particular String is stored.
- Primitive types store values not addresses
  - `x=3;`

References

- Two reference variables can reference the same object
  - `String welcome=greeting;`
  - copies the address in `greeting` to `welcome`
- Creating new objects
  - `String keyboard = new String("qwerty");`

Self-Check

- `String y=new String(“abc”);`
- `String z=“def”;`
- `String w=z;`

Control Statements

- if … else
- switch
- while
- do … while
- for

Examples

- Compute the sum of all even numbers from 2 to 200 unless they are multiples of 7
- Come up with a natural `do … while` example

Calling by Value

- In Java all arguments are call-by-value
  - If the argument is a primitive type, its value, not its address, are passed to the method
  - The method cannot modify the argument value and have this modification remain after returning
  - If the argument is of class type, it can be modified using its own methods and the changes are permanent
- Other languages also support call-by-reference

The `Math` Class

- Collection of useful methods
- All static

```
public class SquareRoots {
    public static void main(String[] args) {
        System.out.println("n \tsquare root");
        for (int n = 1; n <= 10; n++) {
            System.out.println(n + "\t" +
                               Math.sqrt(n));
        }
    }
}
```

The `String` Class

- Assume `keyboard` is a String that contains “qwerty”

```
keyboard.charAt(0)
keyboard.length()
keyboard.indexOf('o')
keyboard.indexOf('y')
String upper=keyboard.toUpperCase();
```

Creates a new string object without changing `keyboard`

Strings are Immutable

- Strings are different from other objects in that they are immutable
  - A String object cannot be modified
  - New Strings are generated when changes are made

```
String myName = "Elliot Koffman";
myName = myName.substring(7) + ", " +
  myName.substring(0, 6);

myName[0]= 'X'; // invalid
myName.charAt(0)= 'X'; // invalid
```

Comparing Objects

```
String anyName = new String(myName);
anyName == myName  ?
```

- `==` operator compares the addresses and not the contents of the objects
- Use `equals, equalsIgnoreCase, compareTo, compareToIgnoreCase`
- Comparison methods need to be implemented for user-defined classes

Wrapper Classes for Primitive Types

- Primitive numeric types are not objects, but sometimes they need to be processed like objects
  - When?
- Java provides *wrapper classes* whose objects contain primitive-type values
  - `Float, Double, Integer, Boolean, Character`
  - They provide constructor methods to create new objects that “wrap” a specified value
  - Also provide methods to “unwrap”

Autoboxing/Unboxing

- Before Java 5.0, int values and Integer objects could not be mixed

```
int n = nInt.intValue();
nInt = new Integer(n++);
```

- Java 5.0 introduced autoboxing/unboxing

```
int n = nInt;
nInt = n++;
```

or

```
nInt++;
```

Examples

```
Integer i1=35;
Integer i2=1234;
Integer i3=i1+i2;
int i2Val=i2++;
int i3Val=Integer.parseInt(“-357”);
Integer i4= Integer.valueOf(753);
System.out.println(i1);
```

More Examples

- `System.out.println(i1+i2);`
- `System.out.println(i1.toString()+i2.toString());`
- Operations are the same as primitive types in current version of java
  - autoboxes before the operator is applied.
- What happens for the == operator?

Practice: Tic Tac Toe

- The 9 cells on the board are numbered
- Read integers from 1 to 9 from text file
  - Each move is given by the cell number the player wishes to occupy
  - Players alternate starting with X, so the ID of the current player is not given
- -1 denotes the end of the game

0	1	2
3	4	5
6	7	8

Requirements

1. If the number is -1, reset the board and start a new game.
2. Check that the move is legal, i.e. the cell is not occupied.
3. Check if there is a winner. There are multiple ways to do this.
4. Check if there is a tie.