Data Structures

CS284

This Semester's Team

Instructor: Eduardo Bonelli

Email: `ebonelli@stevens.edu`

CAs: Nathaniel Burgdorfer, Haralambos Gerogiou, Evan Garrahy, Ryan Mccauley, Alexander Schlumpf, Sean Trinh

Webpage: Canvas

Ask questions!

- Learning goes both ways in this course
- Ask questions in class
- Ask questiones by email
- Seek me out during office hours and...ask questions!
- What was the last question you asked this week?

About this course

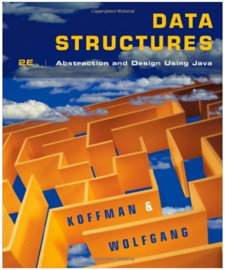
- This is a course on data structures
  - Focus on algorithms
- It is not a course on Java nor object-oriented programming
- We do, however, need a PL in which to put our ideas to work
- That shall be Java
- We could have used others too

Why Java?

- Industry standard (for now)
- Large ecosystem
- Not tied to any particular architecture (Java Virtual Machine)
- Other advantages include security and extensibility

Bibliography

- Intro to Java: Koffman and Wolfgang. Appendix A



- Assignment: Install Eclipse as soon as possible!
- Also install the Java Development Kit

Intro to Java

- We will dedicate the first two weeks to Java
- This is not meant to be an exhaustive coverage
- It is meant to start you off
- You must practice
- Strongly recommended: try out the snippets of code from the slides

Important Information in the Syllabus (Excerpt)

Homework

- Policy for late submissions: 2 points off for every hour past the deadline.
- 0 if code does not compile (.java vs .class)
- 0 if you submit an empty or corrupted archive

Quizzes

- 0 if absent
- Solved in class immediately after handing it in
- You receive two copies of a quiz
  - One copy is handed in (this is not returned)
  - The other copy is for writing down feedback

Exams

- Three
  - Midterm
  - Endterm
  - Final
- The final exam is cumulative.
- Midterm and endterm exam dates are listed in the tentative course schedule available in Canvas.
- If, after the grades for all quizzes, assignments and midterm and endterm are in, your average is 90 or over, you may opt out of the final.

Weight of Grading Categories

Homework	(30%)
Quizzes	(10%)
Midterm	(20%)
Endterm	(20%)
Final Exam	(20%)

Emails

Always:

- Begin your email with a greeting (eg. “Hi”)
- Indicate your class section
- Sign your email with your name

On Slides

- In most lectures I explain by coding directly in Java
  - You are expected to follow my explanations
  - You are not expected to type everything I type myself
  - The code from the lectures will be made available in Canvas after the lecture
- Slides are nevertheless important
  - They contain examples and concepts that are, many times, complementary to the ones I present in class
  - Be sure to read them in your own time

Remaining Slides

What follows marks the first of the set of supporting slides that you are to start reading at your own pace and in your own time

Java Basics

- Classes
- Methods
- An Example

Arrays

More Java

- Type Compatibility and Conversion
- Referencing Objects
- Parameter Passing is Call-by-Value
- More Java Tidbits

Object-Oriented System

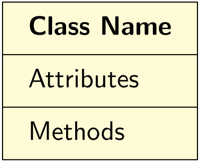
- A set of entities that collaborate with each other in order to perform some specific task
- Entities usually go by the name of objects
- Collaboration is achieved by sending messages from one object to another
- This is one of many models to which a programmer can resort in order to address a (programming) problem
- It is attractive because, in many cases, it reflects rather well the real world entities begin modelled

Java is Object-Oriented

- Java is a PL for implementing object-oriented systems
- A Java program is a collection of classes
- It is based on classes
- A class is a named description for a group of entities that have the same characteristics
  - Entities: Objects or instances of the class
  - Characteristics: attributes (data fields) for each object and the operations (methods) that can be performed on these objects

UML Diagram

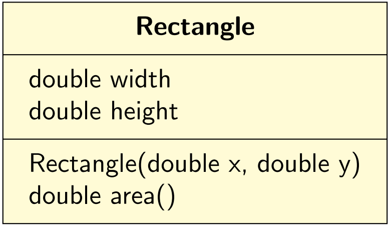
- Graphical representation of classes



Class Name

Attributes

Methods



Rectangle

double width
double height

Rectangle(double x, double y)
double area()

Rectangle Example

- Class definitions in .java files

```java
public class Rectangle{
  // data fields
  private double width;
  private double height;

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

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

Rectangle Example

- Class definitions in .java files

```java
public class Rectangle{
  // data fields
  private double width;
  private double height;

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

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

Rectangle Example

- Class definitions in .java files

```
public class Rectangle{
  // data fields
  private double width;
  private double height;

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

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

Creating Objects Instances of Classes

- Objects may be instantiated from classes using the **new** keyword
- E.g.: **new** Rectangle(3.5, 2.6)
- We can create as many instances as required

```
// text goes in main() method
// create a rectangle with width 3.5 and height 2.6
Rectangle rect1 = new Rectangle(3.5, 2.6);
Rectangle rect2 = new Rectangle(7.2, 8.4);

// get their area
double ar;
ar = rect1.area();
ar = rect2.area();
```

Data Fields and Types

- Data fields are variables
- Variables must be declared with a type before use
- There are primitive data types:

byte	-128 to 127
short	-32,768 to 32,767
int	-2,147,483,648 to 2,147,483,647
long	-2^{63} to $2^{63}-1$
float	32-bit IEEE 754 floating point
double	64-bit IEEE 754 floating point
char	Unicode character set
boolean	true, false

- Special support is provided for strings through the `java.lang.String` class
- Class names are also types (more on this later)

Methods

- A group of statements to perform a particular operation (similar to functions/procedures in other languages)
- Methods are either class or instance methods
  - Instance Methods: Applied to an object using dot notation

    ```
    object.method(arguments)
    ```

    - E.g.

      ```
      rect.area();
      ```
  - Class Methods: Applied to a class using dot notation

    ```
    class.method(arguments)
    ```

    - An example follows

Static Methods

```java
public class Rectangle {
  private double width;
  private double height;
  private static int numberOfRectangles = 0;

  public Rectangle(double x, double y){
    width = x;
    height = y;
    numberOfRectangles++;
  }
  public static int getNumberOfRectangles() {
    return numberOfRectangles;
  }
}
```

Static Methods

```java
public class Rectangle {
  private double width;
  private double height;
  private static int numberOfRectangles = 0;

  public Rectangle(double x, double y){
    width = x;
    height = y;
    numberOfRectangles++;
  }
  public static int getNumberOfRectangles() {
    return numberOfRectangles;
  }
}
```

- `static` indicates that it is a class method
- There is one per class
- Called using dot notation

  ```java
  int i = Rectangle.getNumberOfRectangles();
  ```

- Static methods cannot call instance methods

Static vs Instance Methods

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

}
```

The `main` method

Point where execution begins

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

Eg.

```java
public class Rectangle {
  ...
  public static void main( String[] args){
    Rectangle rect = new Rectangle(3.5, 2.6);
    double ar;
    ar = rect.area();
    System.out.println(ar);
  }
}
```

Java Basics

- Classes
- Methods
- An Example

Arrays

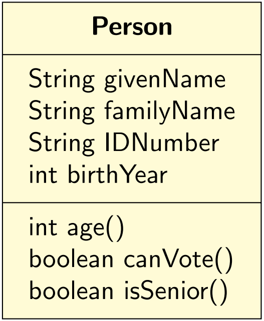
More Java

- Type Compatibility and Conversion
- Referencing Objects
- Parameter Passing is Call-by-Value
- More Java Tidbits

A class `Person`

- Attributes:
  - 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 for Class Person



Person

String givenName
String familyName
String IDNumber
int birthYear

int age()
boolean canVote()
boolean isSenior()

- Style: use of camel notation such as in `myVariable` and `thisLongIdentifier`

Defining the Class Person

```java
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;
```

Comments in code:

`//` vs `/**... */` vs `/* ... */`

Defining the Class Person

```java
// Constants
/** The age at which a person can vote */
private static final int VOTE_AGE = 18;
/** Age at which person considered senior citizen */
private static final int SENIOR_AGE = 65;
```

- Style: Primitive type constants all uppercase

Private Data Fields and Public Methods

- Access modifiers such as **`public`** and **`private`** let you control what other classes have access to a member field
- **`public`**: the field/method is accessible from all classes
- **`private`**: the field/method is accessible only within its own class
- Common to make fields private and methods public
- Details of how data are stored and represented can be changed without affecting class's clients

```java
// 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 bi
  givenName  = first;
  familyName = family;
  IDNumber   = ID;
  birthYear  = birth;
}

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

Constructors

- Four-parameter

```java
public Person(String first, String family, String ID, int
```

- One-parameter

```java
public Person(String ID) {...}
```

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

Instance Methods for Modifying Instance Variables

```
 // 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;
}
```

Use of this

```
  /** Sets the birthYear field.
      @param birthYear The year of birth
   */
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

Sample Instance Methods for Accessing Instance Variables

```java
// 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;
}
```

```java
// Other Methods
/**  Calculates 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;
}
```

The Method `toString`

```java
/** 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";
}
```

- Display the state of author1 (an instance of Person):

```java
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`

```java
  /** 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.getIDNumber());
  }
}
```

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

Testing Class `Person`

```java
public class TestPerson {
 public static void main(String[] args) {
   Person p1 = new Person("Sam", "Jones", "1234", 1930);
   Person p2 = new Person("Sue", "Jones", "5678", 1990);

   System.out.println("Age of " + p1.getGivenName() +
                      " is " + p1.age(2012));

  // prints: Age of Sam is 82

}
```

Testing Class `Person`

```java
public class TestPerson {
 public static void main(String[] args) {
   Person p1 = new Person("Sam", "Jones", "1234", 1930);
   Person p2 = new Person("Sue", "Jones", "5678", 1990);

   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");

// prints: Sam can ride the subway for free


 }
}
```

Testing Class `Person`

```java
public class TestPerson {
 public static void main(String[] args) {
   Person p1 = new Person("Sam", "Jones", "1234", 1930);
   Person p2 = new Person("Sue", "Jones", "5678", 1990);

   System.out.println("Age of " + p2.getGivenName() +
                      " is " + p2.age(2012));

// prints: Age of Sue is 22

   if (p2.canVote(2004))
    System.out.println(p2.getGivenName()+" can vote");
   else
    System.out.println(p2.getGivenName()+" can’t vote");

// prints: Sue can’t vote

 }
}
```

Java Basics

- Classes
- Methods
- An Example

Arrays

More Java

- Type Compatibility and Conversion
- Referencing Objects
- Parameter Passing is Call-by-Value
- More Java Tidbits

Arrays

```java
int[] scores = new int[5];
```

- Declares an array of size 5
- First item starts at index 0
- Arrays are initialized by default in Java
- This prints five zeros

```java
int[] scores    = new int[5];
for (int i=0; i<5; i++) {
  System.out.println(scores[i]);
};
```

Arrays

- We can also initialize the elements with our own values

```java
String[] names = {"Sally", "Jill", "Hal", "Rick"};
System.out.println(names.length);
// length above is data field, not a method
```

- The elements of an array can also have user defined types

```java
Person[] people;
int n       = 3+4;
people      = new Person[n];
people[0] = new Person("Elliot","Koffman","123",1942);
```

Arrays

- There is an enhanced for loop for collections, arrays included
- Rather than

```java
for (int i=0; i<5; i++) {
  System.out.println(scores[i]);
};
```

- We can write

```java
for (int i : scores) {
  System.out.println(scores[i]);
};
```

Two-Dimensional Arrays

```java
final int ROWS = 3;
final int COLS = 3;
double[][] matrix = new double[ROWS][COLS];

for (int i =0; i<ROWS; i++) {
    for (int j=0; j<COLS; j++) {
        System.out.println(matrix[i][j]);
    }
}
```

Java Basics

- Classes
- Methods
- An Example

Arrays

More Java

- Type Compatibility and Conversion
- Referencing Objects
- Parameter Passing is Call-by-Value
- More Java Tidbits

Type Compatiblity 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 = 42;
double realItem = item; // valid ?

double y = 3.14;
int x = y; // valid ?
```

Type Compatiblity 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 = 42;
double realItem = item; // valid ?

double y = 3.14;
int x = y; // valid ?
```

“Type mismatch: cannot convert from double to int”

Type Compatiblity and Conversion

- We can add a type cast to instruct the compiler that `y` should be considered as having type **`int`**

```
double y = 3.14;
int x = (int) y;
```

Referencing Objects

```java
String greeting;
greeting = "hello";
```

- String object “hello” is now referenced by `greeting`
- `greeting` stores the address where a particular String is stored
- Primitive types store values not addresses (Eg. `x=3`)
- Two reference variables can reference the same object

  ```java
  String welcome=greeting;
  ```

- copies the address in greeting to welcome

Referencing Objects – Copying an Array

- Assignment copies only references to objects
- Eg. The following prints 8

```java
int[] data1 = {1,2,3,4,5};
int[] data2 = data1;
data2[0] = 8;
System.out.println(data1[0]);
```

- In order to make a copy of an array we use the `clone` method
- Eg. The following prints 1

```java
int[] data1 = {1,2,3,4,5};
int[] data2 = data1.clone();
data2[0] = 8;
System.out.println(data1[0]);
```

Parameter Passing is Call-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

Parameter Passing is Call-by-Value

```java
public void foo(Dog d) {
    d = new Dog("Snoopy"); // creates the "Snoopy" dog
}

Dog aDog = new Dog("Pluto"); // creates the "Pluto" dog
// aDog points to the "Pluto" dog
foo(aDog);
// aDog still points to the "Pluto" dog
```

The `Math` Class

- Collection of useful methods
- All static

```java
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) // q
keyboard.length()  // 6
keyboard.indexOf(’o’) // -1
keyboard.indexOf(’y’) // 5
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, String is not an Array
myName.charAt(0)= ’X’; // invalid
```

Comparing Objects

```java
String myName = "Elliot Koffman";
String anyName = new String(myName);
System.out.println(anyName == myName); // false
System.out.println(anyName.equals(myName)); // true
```

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

Wrapper Class for Primitive Types

- Primitive numeric types are not objects, but sometimes they need to be processed like objects
- Eg. When primitive types must be inserted into collections
- Java provides wrapper classes whose objects contain primitive-type values

`byte`	`Byte`	**`float`**	`Float`
`boolean`	`Boolean`	**`int`**	`Integer`
`char`	`Character`	**`long`**	`Long`
`double`	`Double`	**`short`**	`Short`

- They provide constructor methods to create new objects that “wrap” a specified value and methods to “unwrap”
- This is typically done automatically in most cases (process known as autoboxing)