Final Exam Review

Final exam information

- Date: Friday, June 26
- Coverage: Chap 1 – 11 and 16 (open book)
- 7 problems, 3 hours
- format similar to our previous exams
- 50% of the total grade

Final exam format

- Prob. 1 – short answers
- Prob. 2 – simple statements and code
- Prob. 3, 4, 5 – programming problems
  - writing static methods
  - arrays, array lists, etc.
  - iteration, search, compare, count …
- Prob. 6 – implement a class
- Prob. 7 – inheritance, polymorphism, interface

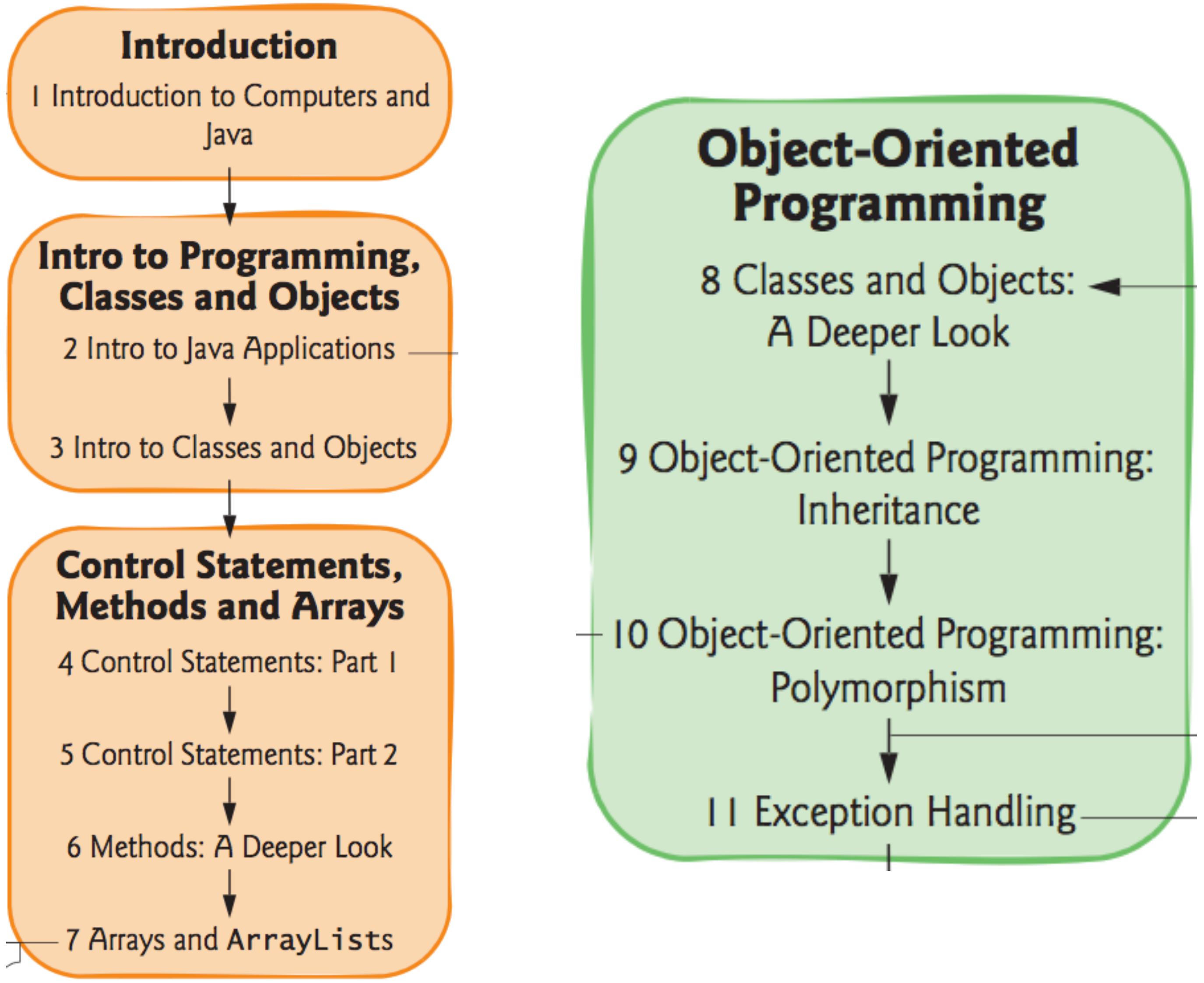
Book chapters

1. Intro to Computers and Java
2. Intro to Java Applications
3. Intro to Classes, Objects, Methods and Strings
4. Control Statements: Part 1
5. Control Statements: Part 2
6. Methods: A Deeper Look
7. Arrays and ArrayLists
8. Classes and Objects: A Deeper Look
9. Object-Oriented Programming: Inheritance
10. Object-Oriented Programming: Polymorphism
11. Exception Handling: A Deeper Look
16. Generic Collections (List, Set, Map)

Course syllabus

- Compiling and running Java programs
- Java packages
- Selection and iteration
- Implementing methods
- Arrays (single and two dimensions)
- Array lists (dynamic arrays)
- Classes and objects
- Extending classes with inheritance
- Abstraction, inheritance, and interfaces
- Exceptions
- Graphics and event-driven programs
- Extra: File and image I/O, Swing GUI, little bit of Android

Main chapters and modules


Introduction
1 Introduction to Computers and Java
Intro to Programming, Classes and Objects
2 Intro to Java Applications
3 Intro to Classes and Objects
Control Statements, Methods and Arrays
4 Control Statements: Part 1
5 Control Statements: Part 2
6 Methods: A Deeper Look
7 Arrays and ArrayLists
Object-Oriented Programming
8 Classes and Objects: A Deeper Look
9 Object-Oriented Programming: Inheritance
10 Object-Oriented Programming: Polymorphism
11 Exception Handling

Practice Problems

Problem: Short answers

a) What is an *abstract class* in Java?

An abstract class defines one or more *abstract* methods whose implementation is provided in its *concrete* subclasses.

b) True / False: *Polymorphism* is the ability to inherit functionality from other classes.

False. Polymorphism is the ability for various classes to respond and behave as if belonging to the same type.

Problem: Simple statements

a) Declare and initialize an array object dblArray consisting of 16 double values.

```
double[] dblArray = new double[16];
```

b) Declare and initialize an array list flagList that can hold any number of Boolean values.

```
ArrayList<Boolean> flagList = new ArrayList<Boolean>();
```

Problem: Return diagonal product

Implement a static method `diagProd`, which returns the product of the diagonal entries of a square matrix.

Use the following method prototype, where matrix is an input integer square matrix:

```
public static int diagProd( int[][] matrix )
```

Solution:

```
public static int diagProd( int[][] matrix )
{
    int prod = 1;

    for( int i=0; i < matrix.length; i++ ) {
        prod *= matrix[i][i];
    }

    return prod;
}
```

Problem: Count filtered matrix elements

Implement a static method `countGreaterThan`, which counts the elements of a square matrix that are greater than an input threshold value.

Use the following method prototype, where matrix is an input integer square matrix:

```
                    public static int
    countGreaterThan( int[][] matrix, int val )
```

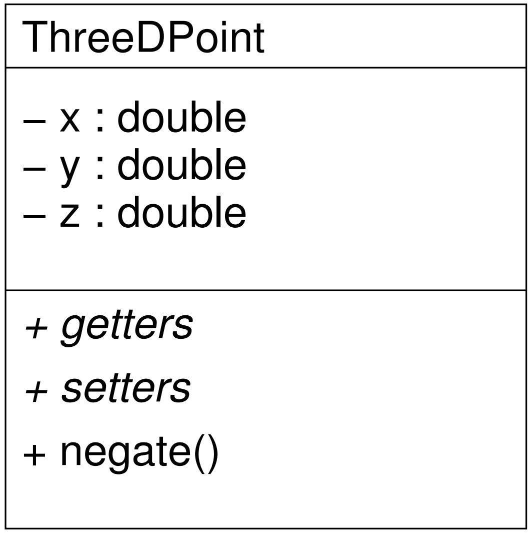
Solution algorithm:

```
int count = 0;

for( int i=0; i < matrix.length; i++ ) {
      for( int j=0; j < matrix[i].length; j++) {
            if( matrix[i][j] > val ) {
                  count++;
            }
      }
}

return count;
```

Problem: Implement a class

Implement a simple `ThreeDPoint` class that models points in 3D vector space (each point has three coordinates x, y, and z, stored as private double variables).

The UML diagram for the class is shown below.



Solution:

```
class ThreeDPoint {

    private double x;
    private double y;
    private double z;

    public ThreeDPoint() {
        x = 0.0;
        y = 0.0;
        z = 0.0;
    }

    public double getX() {
        return x;
    }

    public void setX(double x) {
        this.x = x;
    }
```

Solution (cont.)

```
class ThreeDPoint {

    public void negate() {
        x = -x;
        y = -y;
        z = -z;

        // or multiply by -1:
        // x *= -1;
        // y *= -1;
        // z *= -1;
    }

    public String toString() {
        return "(" + x + "," + y + "," + z + ")";
    }

}
```

Problem: Polymorphism with Actors

For definition of the `Actor` interface and the `Comedian` and `ActionHero` classes, refer to the practice final exam.

Write a main method that creates an `Actor[]` array named `myMovieCast`, with three elements and populate it with one comedian and two action heroes. Then iterate over each member of the array, and print out the string output of its `act` method call.

Solution:

```
Actor[] myMovieCast = new Actor[3];

myMovieCast[0] = new Comedian();
myMovieCast[1] = new ActionHero();
myMovieCast[2] = new ActionHero();

for(Actor myActor : myMovieCast) {
      System.out.println(myActor.act());
}
```

Good luck!