CS121 Data Structures A, C
Introduction

Varduhi Yeghiazaryan
vyeghiazaryan@aua.am

Fall 2021

Course Structure, Section A

Timetable (subject to small changes)

classes	**314W**	**Tue, Thu 09.00–10.15**
office hours	**331W**	**Tue, Fri 13.30–14.30 or by appointment**
problem solving sessions	**TBA**	**Wed 15.30–17.30**
TA office hours	**TBA**	**Fri 16.30–18.00**

Prerequisites

- CS120 Introduction to Object-Oriented Programming
- CS111 Discrete Mathematics

If unsure, talk to me right after this class!

Course Structure, Section C

Timetable (subject to small changes)

classes	**314W**	**Tue, Thu 10.30–11.45**
office hours	**331W**	**Tue, Fri 14.30–15.30 or by appointment**
problem solving sessions	**TBA**	**Tue 15.30–17.30**
TA office hours	**TBA**	**Wed 13.00–14.30**

Prerequisites

- CS120 Introduction to Object-Oriented Programming
- CS111 Discrete Mathematics

If unsure, talk to me right after this class!

Textbooks

Main textbook:
Data Structures and Algorithms in Java, 6th edition, by M.T. Goodrich, R. Tamassia, M.H. Goldwasser. John Wiley & Sons, 2014.

Alternative:
Data Structures and Algorithms in C++, 2nd edition, by M.T. Goodrich, R. Tamassia, D.M. Mount. John Wiley & Sons, 2011.

Additional/reference:
Data Abstraction and Problem Solving with Java/C++, by F.M. Carrano, J.J. Prichard, T.M. Henry

Additional materials may be posted on Moodle

Assessment

Homework ($\times 7$)	18%	Released bi-weekly
Pop Quizzes ($\times 5$)	10%	15–20 mins each, lowest dropped
Midterms ($\times 2$)	34%	Sat, Oct 9 and Nov 20
Final Exam	38%	

Exact dates and further details can be found in the Syllabus

Homework Rules

Late submission is **not** graded; you can still submit to get feedback

Any collaboration or usage of materials (e.g. online sources) should be **explicitly acknowledged**. Acceptable for groups of **max. 2** people. The assignment is graded at **70%** of the actual score.

First unacknowledged collaboration or usage of materials: the **whole** assignment graded **zero**
Second occurrence of cheating: a **zero** grade for all homework assignments, i.e. 18% for the course grade
Third cheating attempt: an **F** grade for the course

Short feedback of a few sentences will be provided on Moodle

You **should discuss** your work in more detail with the TA during the **office hours** following the submission deadline

The **TAs will collaborate** to check and grade your homework

Grade Mapping

Grade	Grade Point	Percentile
A+	4	$[95, 100]$
A	4	$[90, 95)$
A−	3.7	$[85, 90)$
B+	3.3	$[80, 85)$
B	3	$[75, 80)$
B−	2.7	$[70, 75)$
C+	2.3	$[66, 70)$
C	2	$[62, 66)$
C−	1.7	$[58, 62)$
D+	1.3	$[55, 58)$
D	1	$[53, 55)$
D−	0.7	$[50, 53)$
F	0	$[0, 50)$

How to Succeed in This Course?

- Work regularly!
  The course is very intensive and incremental, with new topics introduced at each class.
- Do the reading! As much as you can!
  Classes cannot cover all the details. Reading the materials ensures better understanding of the topics.
- Code, code, and code again!
  Mastery comes with experience. The more you do coding the more speed and accuracy you develop. Solving problems using the new algorithms enhances understanding of the material.
- Attend (and arrive on time for) all the sessions!
  This includes classes, problem solving sessions, office hours.
- Do all the assignments!
  If you skip assignments, problems will arise later in the course.
- Ask questions and report issues!
  Make good use of office hours to do this as soon as possible.

Syllabus and Moodle

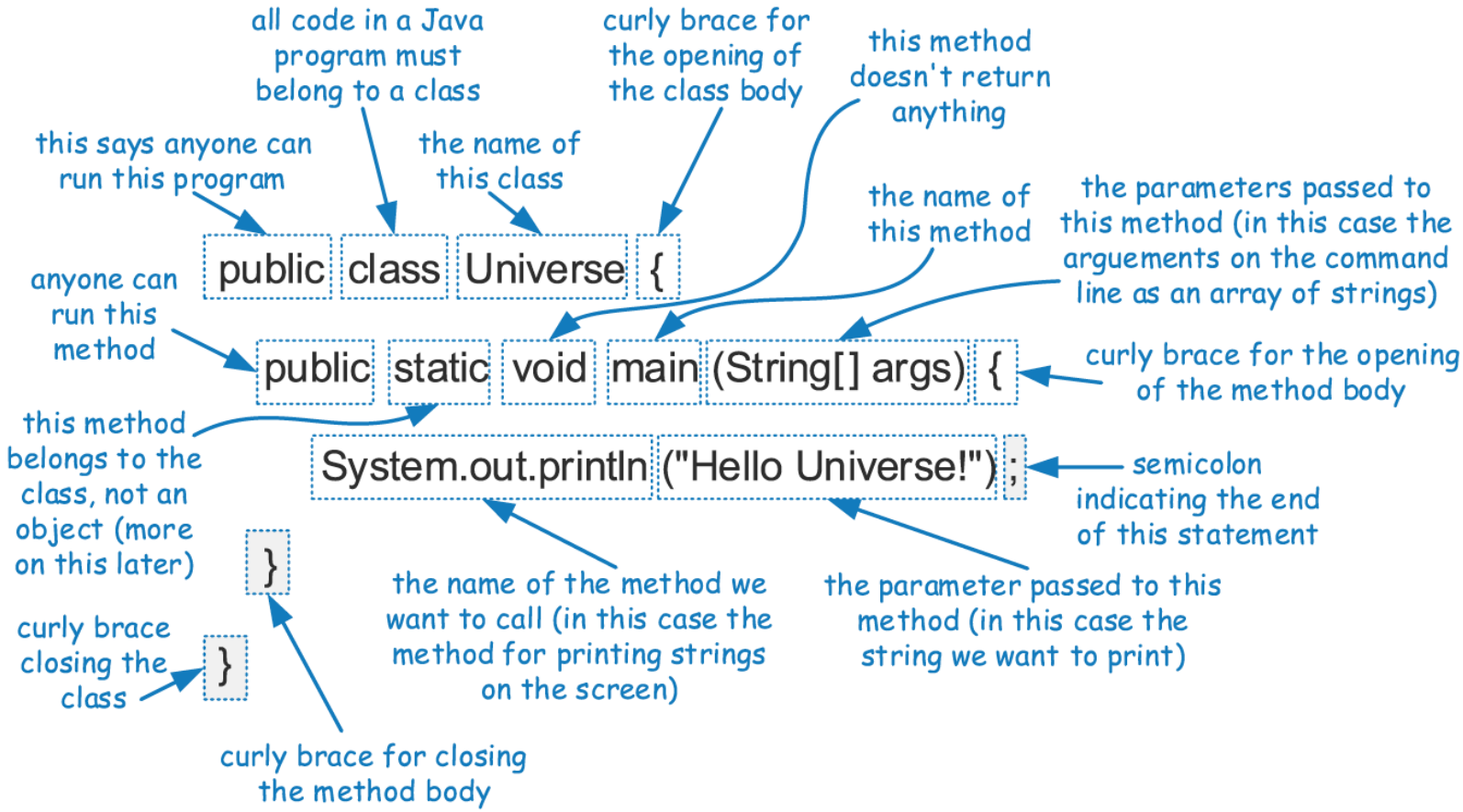
The syllabus and other materials will be available on Moodle

Enrol on Moodle with the key provided today

The course is divided into three main parts:

- complexity, recursion, search, and sorting ($\sim$ 3.5 weeks)
- linear data structures ($\sim$ 4 weeks)
- non-linear data structures ($\sim$ 7.5 weeks)

Simple Program in Java



Base Types

Programming languages typically have several base types, which are basic ways of storing data

An identifier variable can be declared to hold any base type and it can later be reassigned to hold another value of the same type

boolean	a boolean value: true or false
char	16-bit Unicode character
byte	8-bit signed two's complement integer
short	16-bit signed two's complement integer
int	32-bit signed two's complement integer
long	64-bit signed two's complement integer
float	32-bit floating-point number (IEEE 754-1985)
double	64-bit floating-point number (IEEE 754-1985)

```
boolean flag = true;
boolean verbose, debug;
char grade = 'A';
byte b = 12;
short s = 24;
int i, j, k = 257;
long l = 890L;
float pi = 3.1416F;
double e = 2.71828, a = 6.022e23;
```

Class Types

Every **object** is an instance of a **class**, which serves as the type of the object and as a blueprint, defining the **data** which the object stores (**instance variables, or fields**) and the **methods** for accessing and modifying that data.

```
public class Counter {
  private int count;                                  // a simple integer instance variable
  public Counter( ) { }                               // default constructor (count is 0)
  public Counter(int initial) { count = initial; }          // an alternate constructor
  public int getCount( ) { return count; }                   // an accessor method
  public void increment( ) { count++; }                       // an update method
  public void increment(int delta) { count += delta; }        // an update method
  public void reset( ) { count = 0; }                         // an update method
}
```

Class Example

```
public class CounterDemo {
  public static void main(String[ ] args) {
    Counter c;                          // declares a variable; no counter yet constructed
    c = new Counter( );                 // constructs a counter; assigns its reference to c
    c.increment( );                     // increases its value by one
    c.increment(3);                     // increases its value by three more
    int temp = c.getCount( );           // will be 4
    c.reset( );                         // value becomes 0
    Counter d = new Counter(5);         // declares and constructs a counter having value 5
    d.increment( );                     // value becomes 6
    Counter e = d;                      // assigns e to reference the same object as d
    temp = e.getCount( );               // will be 6 (as e and d reference the same counter)
    e.increment(2);                     // value of e (also known as d) becomes 8
  }
}
```

Data Structures

A **data structure** is a

- data organization,
- management and
- storage format

that enables efficient access and modification

It is a *collection* of data values, the *relationships* among them, and the *functions or operations* that can be applied to the data

Simple examples: arrays; box with coloured balls

Abstract Data Types

Abstraction is to distill a system to its most fundamental parts

Applying the abstraction paradigm to the design of data structures gives rise to **abstract data types (ADTs)**

An ADT is a model of a data structure that specifies the **type** of data stored, the **operations** supported on them, and the **types of parameters** of the operations

An ADT specifies *what* each operation does, but not *how* it does it

The collective set of behaviours supported by an ADT is its **public interface**

Sample Program in Java: 1

```
1  public class CreditCard {
2    // Instance variables:
3    private String customer;          // name of the customer (e.g., "John Bowman")
4    private String bank;              // name of the bank (e.g., "California Savings")
5    private String account;           // account identifier (e.g., "5391 0375 9387 5309")
6    private int limit;                // credit limit (measured in dollars)
7    protected double balance;         // current balance (measured in dollars)
8    // Constructors:
9    public CreditCard(String cust, String bk, String acnt, int lim, double initialBal) {
10     customer = cust;
11     bank = bk;
12     account = acnt;
13     limit = lim;
14     balance = initialBal;
15   }
16   public CreditCard(String cust, String bk, String acnt, int lim) {
17     this(cust, bk, acnt, lim, 0.0);                  // use a balance of zero as default
18   }
```

Sample Program in Java: 2

```
19    // Accessor methods:
20    public String getCustomer( ) { return customer; }
21    public String getBank( ) { return bank; }
22    public String getAccount( ) { return account; }
23    public int getLimit( ) { return limit; }
24    public double getBalance( ) { return balance; }
25    // Update methods:
26    public boolean charge(double price) {          // make a charge
27      if (price + balance > limit)                 // if charge would surpass limit
28        return false;                              // refuse the charge
29      // at this point, the charge is successful
30      balance += price;                            // update the balance
31      return true;                                 // announce the good news
32    }
33    public void makePayment(double amount) {       // make a payment
34      balance -= amount;
35    }
36    // Utility method to print a card's information
37    public static void printSummary(CreditCard card) {
38      System.out.println("Customer = " + card.customer);
39      System.out.println("Bank = " + card.bank);
40      System.out.println("Account = " + card.account);
41      System.out.println("Balance = " + card.balance);   // implicit cast
42      System.out.println("Limit = " + card.limit);       // implicit cast
43    }
44    // main method shown on next page...
45  }
```

Sample Program in Java: 3

```
1   public static void main(String[ ] args) {
2     CreditCard[ ] wallet = new CreditCard[3];
3     wallet[0] = new CreditCard("John Bowman", "California Savings",
4                                "5391 0375 9387 5309", 5000);
5     wallet[1] = new CreditCard("John Bowman", "California Federal",
6                                "3485 0399 3395 1954", 3500);
7     wallet[2] = new CreditCard("John Bowman", "California Finance",
8                                "5391 0375 9387 5309", 2500, 300);
9
10    for (int val = 1; val <= 16; val++) {
11      wallet[0].charge(3*val);
12      wallet[1].charge(2*val);
13      wallet[2].charge(val);
14    }
15
16    for (CreditCard card : wallet) {
17      CreditCard.printSummary(card);          // calling static method
18      while (card.getBalance( ) > 200.0) {
19        card.makePayment(200);
20        System.out.println("New balance = " + card.getBalance( ));
21      }
22    }
23  }
```

Summary

Reading

Java language review: Chapter 1 Java Primer

OOP review: Chapter 2 Object-Oriented Design

Questions?