CS310 - Advanced Data Structures and Algorithms

Spring 2014 – Class 1

January 28, 2014

Contact Information

- Instructor: Nurit Haspel
- `http://www.cs.umb.edu/~nurith`
- nurith@cs.umb.edu or nurit.haspel@umb.edu
- Phone – 617-287-6414.
- Office – S-3-071
- Office hours - Tu Th 3:00–4:00 or by appointment.
- Course schedule: Tu Th 5:30–6:45, M-1-614

Course Description

- Roughly three parts:
- Part 1:
  - Methods for structuring and manipulating data in computing.
  - Application program interface (API), data abstraction and encapsulation.
- Part 2: Design and analysis of algorithms, including theoretical background.
- Part 3: Graph theory and applications.
- Throughout the entire course we will study advanced techniques for program development and organization.
- Course website: `http://www.cs.umb.edu/cs310`.
- http://www.cs.umb.edu/cs310/Syllabus.html.

General Stuff

- The course material will be available online and updated regularly with class notes and assignments.
- Attendance is not required (but highly encouraged). **You are responsible for keeping yourselves up to date if you miss a class**.
- Don't be afraid to ask questions in or out of class. I highly encourage it, I won't think you are stupid and it won't lower your grade.
- Don't hesitate to send me e-mails. I expect e-mails. It won't lower your grade.
- In your e-mails be as specific and give as many details as possible.
- However, don't expect me to solve your homework for you or debug your code.

Course Requirements

- Prerequisite: CS210, CS240, MATH140 or equivalent.
- Qualifying exam due Thursday in class.
- Homework (theoretic) and programming assignments approx. every week (15% each).
- Midterm exam (30%), final exam (40%).
- First homework posted online. Due next Tuesday.

Course Requirements (Cont.)

- Your final grade should be at least D- (at least 40) to pass.
- You also have to pass the final exam.
- Book – Weiss, Data Structures and Problem Solving Using Java, 4^{th} edition.
- You may use the 3^{rd} edition at your own risk!
- Grader – Binh Tran. e-mail: tdbinh@gmail.com .

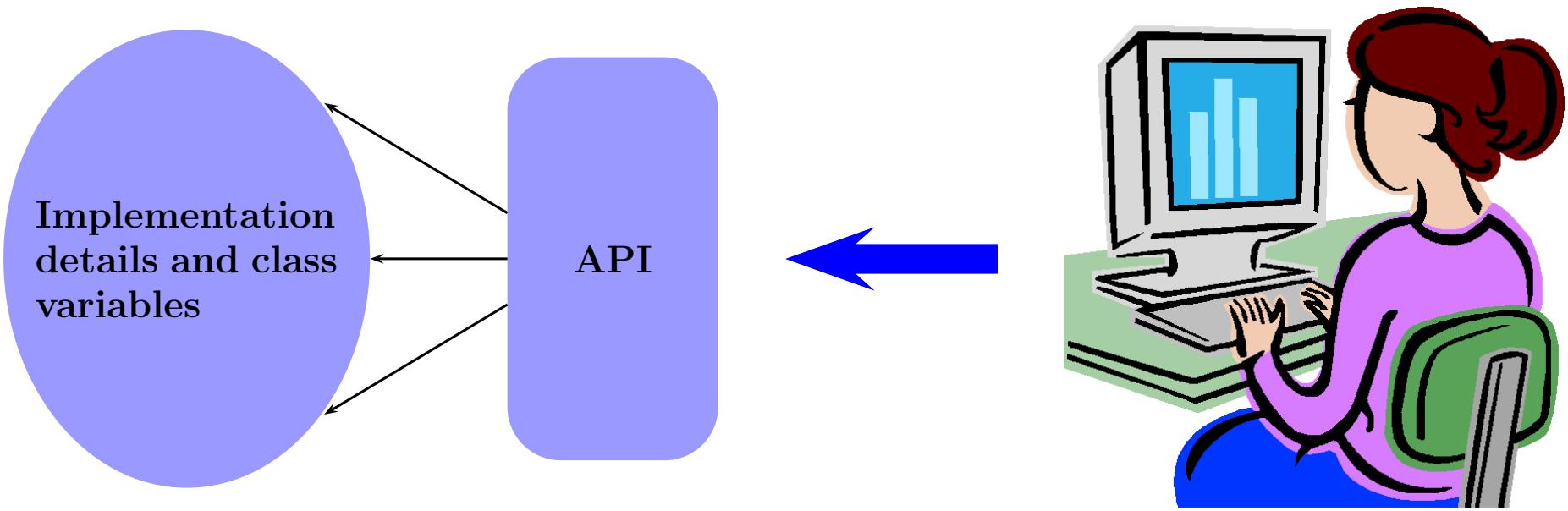
Course Requirements (Cont.)

- Apply for an account with the course ASAP.
- You will join the mailing list and get a subdirectory where pa's can be submitted.
- Read chapters 1-4, know chapters 7,8,16,17. We'll start with chapter 5.
- Do the Qual – due next Thursday.
- If you have a problem solving the qualifying exam - talk to me.

API and Encapsulation

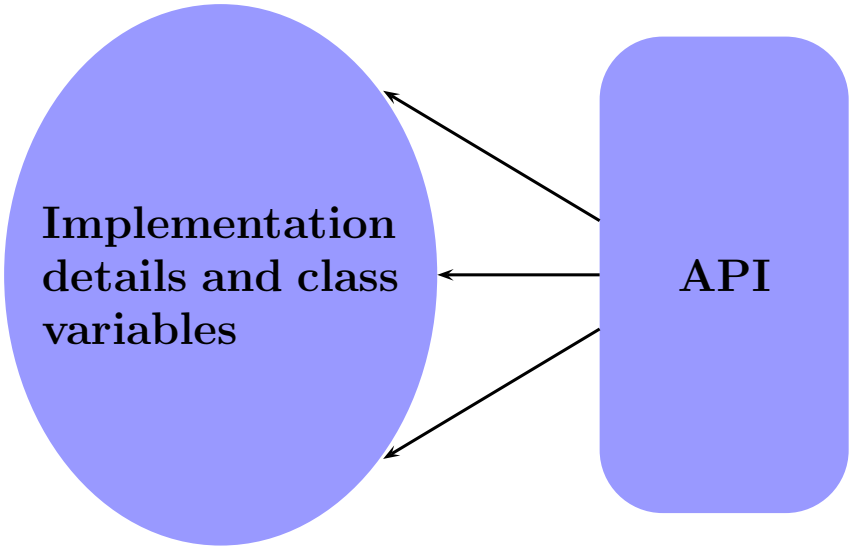
- **Application Programming Interface (API):** a group of function specifications that are meant to provide some service or services to programs, clients of the API.
- **Data encapsulation:** Hiding the data of the implementation of the API functions so well that the client cant get at it by normal programming methods.
- They **have** to use the API functions to get their work done.

API – application programming interface



Example: C string library functions: strcpy, strcmp
Example: The Person class in Weiss, p102: getName(), setPhoneNumber()...

API and Encapsulation



Bank Account Example

- Example – bank account.
- Construct an account:
  `BankAccnt ba = new BankAccnt(‘‘JJ’’, 234, 100);`
- Modify balance:
  `ba.withdraw(20); ba.deposit(10);`
- Access balance:
  `ba.balance; // Incorrect!!`
  `ba.getBalance(); // Correct!!`
- The API is the method specification. An API lies between two bodies of code, the client code and the class implementation.

Bank Account Example

Java interface for clear implementation:

```
public interface Account {
 // withdraw amt from this BA
 int withdraw(int amount);
 // deposit amt to this BA
 void deposit(int amount);
 // return balance for this BA
 int getBalance();
}
```

Bank Account Example

Any class that has balance going up and down can implement this interface:

```
public class BankAccount implements Account {
 // constructor - create a BA
 public BankAccount(String nm, int _id, int bal)
{...}
 // Account API functions
 public int withdraw(int amt) {...}
 public void deposit(int amt) {...}
 public int getBalance() {...}
 // Fields - all private
 private int id;
 private String name;
 private int balance;
}
```

Bank Account Example

Client code:

```
// client code example:
public class TestBankAccount {
 static void main(String [] args)
 {
  BankAccount ba = BankAccount(``JJ'', 234, 100);
  ba.withdraw(20);
  ba.deposit(10);
 }
}
```

Encapsulation in C, the Bigger Challenge

- structs instead of classes.
- Of course all the members are public, so any code can access struct members.
- C does provide **data hiding within source files** through use of static variables.
- See Section 4.6 of K&R for the basics.
- A static variable is hidden from code in other .c files.

Encapsulation in C, the Bigger Challenge

- Hide the members of a struct by making the client code use only **pointers** to BankAccount structs, and not letting the client code see the full struct declaration,
- Example: two .c files both including one **header file providing the incomplete struct definition** along with the API, and possibly some constants.
- One .c file is the implementation and the other holds the client code.

C Bank Account Example

- The bankaccnt struct type is declared by the implementation and understood by it **ONLY**.
- All operations on a BankAccnt instance in Java or C occur via the API.
- A sealed-up C struct instance is very much the same as an object, so we will often use "object" for it, even though officially C has no objects.

Account.h Code

```
/* account.h, header for struct bankacct data type */
/* this allows clients to use struct bankacct pointers */
/* without access to their internals */

struct bankacct; /* incomplete struct declaration */

#define NAMELEN 100 /* max name length allowed */

/* API (int return values provide error codes if necessary, or 0 for
success */
/* Create an account */
struct bankacct *makeBankAccnt (char *name, int id, int bal);
/* Withdraw from account */
int withdraw(struct bankacct *ba, int amt);
/* Deposit amount to account */
int deposit(struct bankacct *ba, int amt);
/* Explicit destruction to free memory, needed in C */
int destroyBA(struct bankacct *ba);
/* return balance for BA ba */
int getbalance(const struct bankacct *ba);

/*(also, getid, getname) */
```

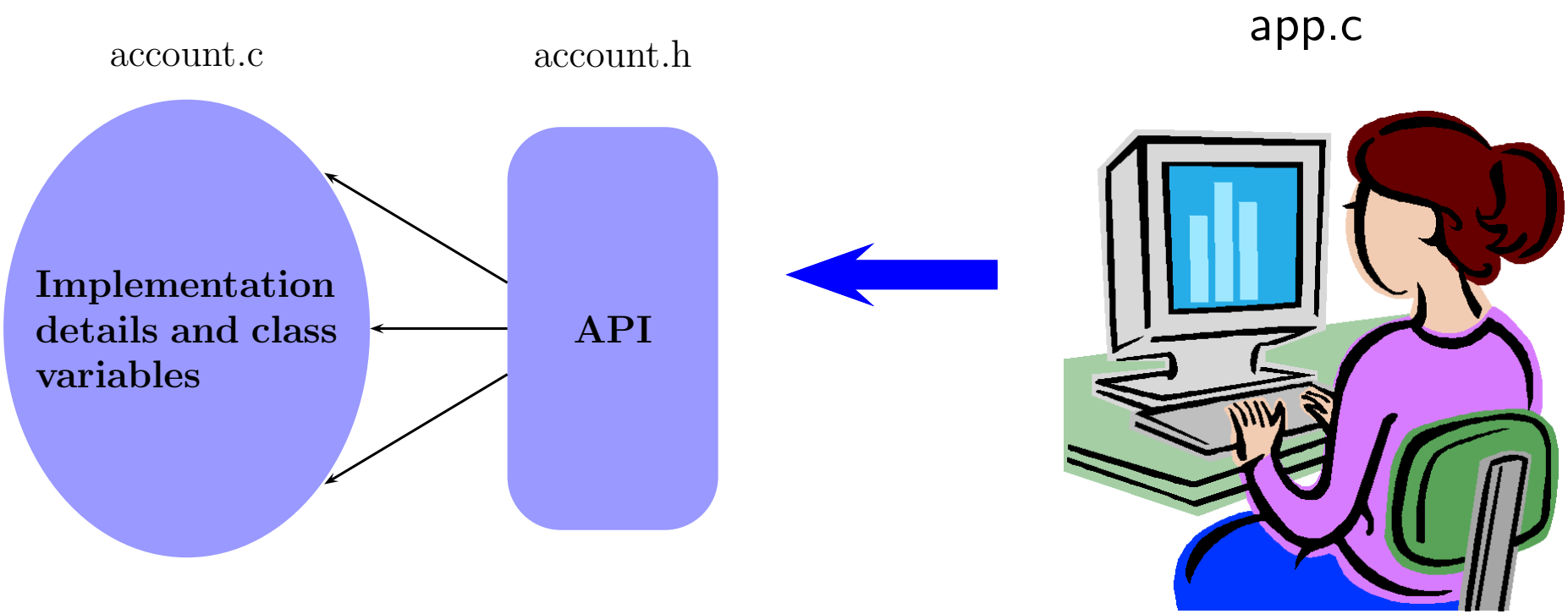
Account.c Code

```
/* account.c, implementation of bankacct data type */
#include ‘‘account.h’’
/* fill in the incomplete struct definition, inside this file */
typedef struct bankacct {
  int id;
  char name[NAMELEN+1];
  int balance;
} BankAccnt;

/* bankacct implementation code */
/* create a BA - allocate memory and assign fields*/
struct bankacct *makeBankAccnt (char *nm, int _id, int bal) {...}
/* withdraw amt from BA ba */ int withdraw(struct bankacct *ba, int amt)
{...}
/* deposit amt to BA ba */
int deposit(struct bankacct *ba, int amt) {...}
/* return balance for BA ba */ int getbalance(const struct bankacct *ba)
{ return ba->balance; }
/*(also, getid, getname, destroyBA)*/
/* Explicit destruction to free memory, needed in C */

int destroyBA(struct bankacct *ba) {...}
```

Client Code

```
 /* client code example:  app.c */
#include ‘‘account.h’’
int main()
{
  struct bankacct *ba =
  makeBankAccnt (‘‘JJ’’, 234, 100);
  withdraw(ba, 20);
  deposit(ba, 10);
  /* in C, we need explicit destruction */
  destroyBankAccnt(ba);
  return 0;
}
```

API – application programming interface



Advantages of Encapsulation

- Clear statement of functionality in use – what is intended/provided by the class.
- Partition of responsibility/code. Important if many programmers.
- Can share General Purpose objects across many apps. Saves coding. At worst have to make minor improvements later.
- Safety of contents. Primitive functions can check arguments, etc.,
- Debugability. No mystery changes to data – can breakpoint the functions that change data.

Qualifying Exam – Background

- **Idea:** 500 PC clients connected to a server (say, with some filesystem mounted on them and located on the server).
- Our system outputs to a log the PC's in use and the user names that are and logged on them.
- Here is part of the log:

9	ALTEREGO
12	ALIMONY
433	HOTTIPS
433	USERMGR
12	BLONDIE
433	HOTTIPS
354	ALIMONY
...	...

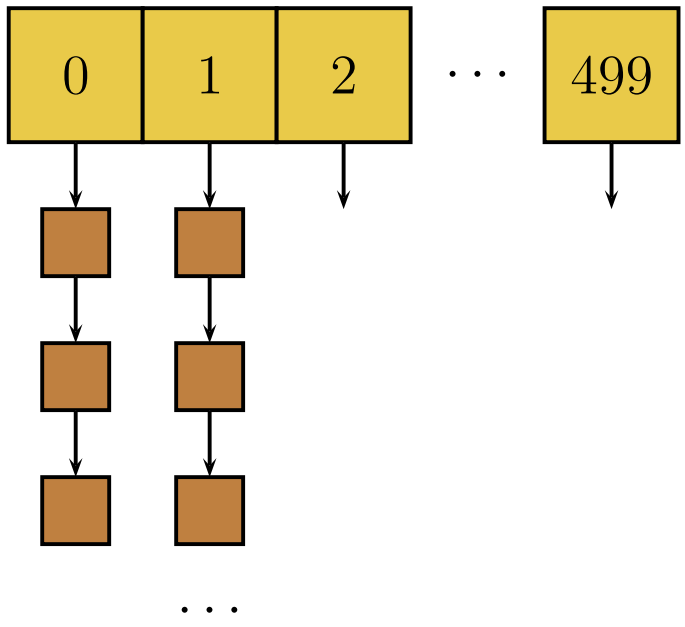
Qualifying Exam – Background

- **Task:** Write a Java program to read all information in from the file (access the file as stdin).
- Then print out report:

Line	Most Common User	Count
1	OPERATOR	174983
2	HANNIBAL	432
3	<NONE>	0
4	SYSMGR	945
...	...	...

Qualifying Exam – Background

- As information is read in we record it in an array of 500 chain linked lists, one for each PC station. We have list 0, 1, . . ., 499.
- Each list is based in an array entry something[i], and holds all the needed information for line i.
- Each node of the list corresponds to one username, and a count of times the username was encountered on this line (and a pointer to the next node).



Qualifying Exam – Background

- Class SinglyLinkedList (SLL) is provided.
- It can add elements to the list via add(), and supports indexed access with get(i).
- Look at main: a test program.
- You should wrap it to make a special-purpose container for a single PC unit called LineUsageData.

Qualifying Exam – Background

What you need to implement:

- class LineUsageData:
  - HAS A SinglyLinkedList<someListElementType>
  - It has a field of type SLL, so it has the use of the SLL to do its own work.
- LineUsageData methods:
  - void addObservation(String userName);
  - Usage getMaxUsage();

Qualifying Exam – Background

- Top level code: TermReport.java: set up a master array of LineUsageData objects, one for each line number.
- Create LineUsageData objects for all elements of the master array.
- Read in data from standard input and access entry in array by line number, then do "addObservation" to that LineUsageData.
- Inside LineUsageData, addObservation searches list for name, when finds name, increments count, else adds new Usage object to SLL.

Qualifying Exam – Background

- When all data has been read in, time for report.
  - Loop through all objects in array.
  - For each, call getMaxUsage and print out line of report.
- Inside LineUsageData, loop through list and remember Node pointer and count for max.
- Here the SLL is a collection class, though not a JDK Collection class.