Computer Systems Design and Applications

CS209A

Lecture 1

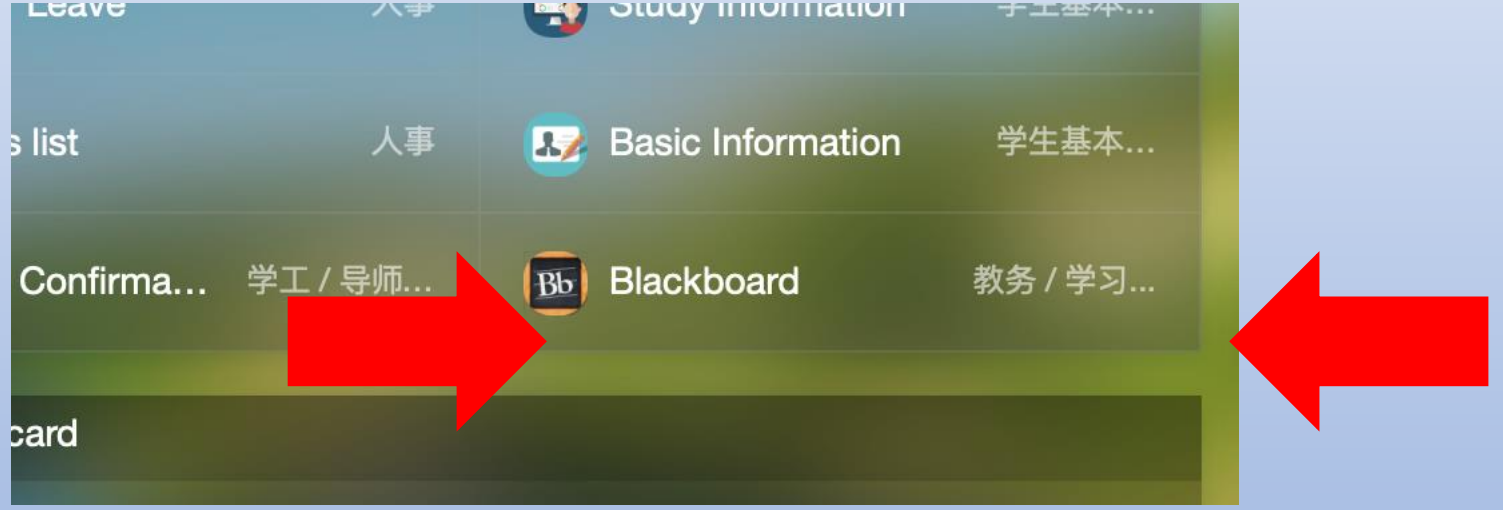
Adam Ghandar

Course Staff

- Lecturer: Adam Ghandar
  - aghandar@sustech.edu.cn
- Tutor: 赵耀(ZHAO Yao)老师
  - zhaoy6@sustech.edu.cn
- Course structure
  - Lectures
  - Labs

Course website

- “Blackboard”
  http://bb.sustech.edu.cn or via Ehall http://ehall.sustech.edu.cn/new/index.html



Assessment

- Participation: 10%
- Project 20% *(release in approx. week 6)*
- Presentation 10% *(about project)*
- Pracs: 10% *(1 mark for each completed successfully up to a max of 10)*
- Assignments 10% *(2 assignments)*
- Mid term 10%
- Final exam 30%

Exam

OPEN BOOK AND
UNLIMITED NOTES

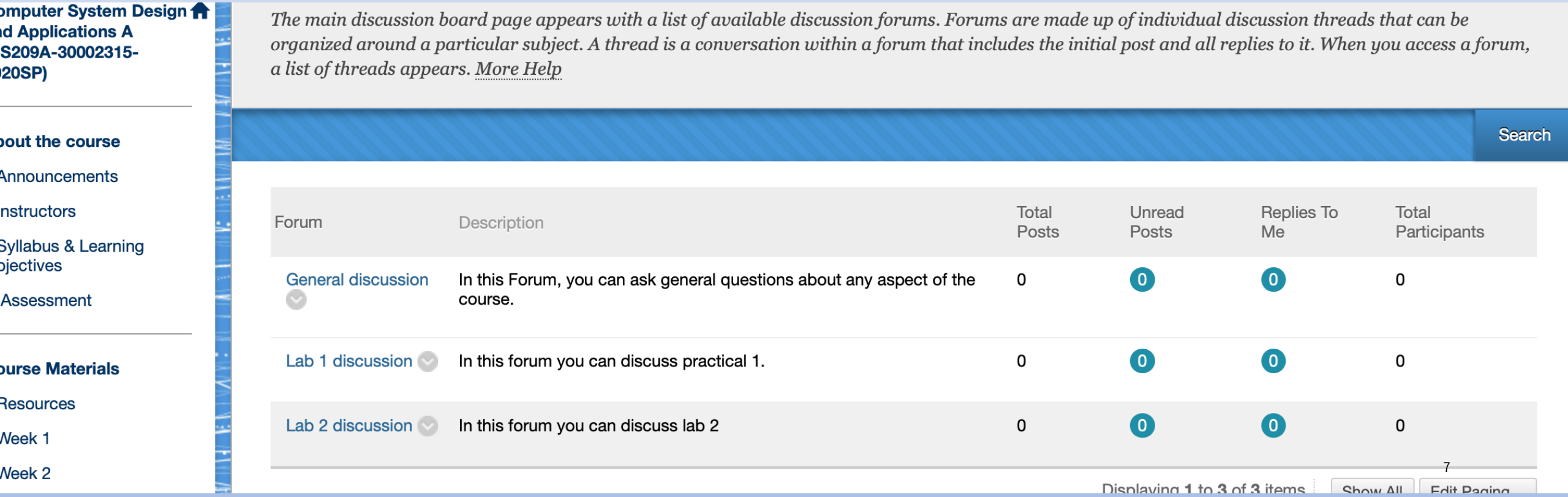
ELECTRONIC DEVICES
FORBIDDEN

Grades

- Your marks can be checked online
- Any queries should be directed to the Lecturer via email clearly stating your student id the assignment and question
- Extensions or requests for special consideration should where possible be supported by documentation and submitted in advance
- Participation will be graded by looking at your posts in the forum and attendance

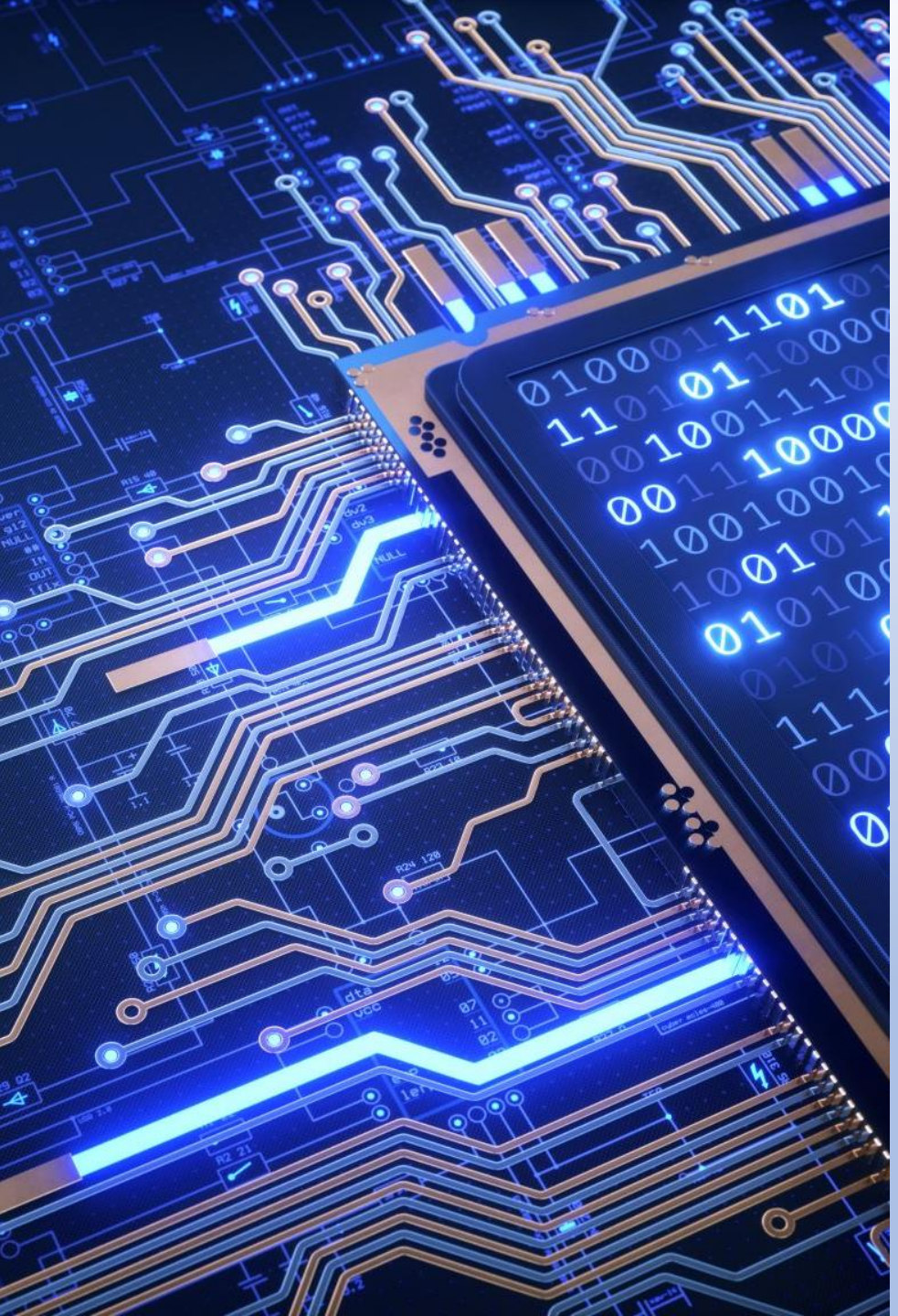
Online Course Forums

- All questions and discussion about the course work can be undertaken here



Course Objectives

This course aims to provide a deeper understanding of programming and new topics in computer application system design, especially data processing, GUI implementations, stream processing, program evaluation, regexp application and other advanced programming topics & skills useful for scientific & engineering students. You will learn how to use java programming to develop software that can solve practical problems.



Outcomes

- On completion of this course, students should be able to:
  - Understand design principles and good practices in software application design.
  - Apply programming concepts from object-oriented programming.
  - Use Java to solve real world problems effectively and efficiently.

Topics Covered

1. Information Systems
2. Data and IO
3. Design paradigms, OOP, practical programming in Java
4. Collections and generic types
5. Advanced programming techniques (nested classes, anonymous classes, inheritance and polymorphism, Streams, regex, reflection and more)
6. Graphical User Interfaces (GUI) with JavaFX
7. Client server applications
8. Multithreading
9. Data persistence including databases
10. Java Virtual Machine (JVM)
11. Testing and test driven development

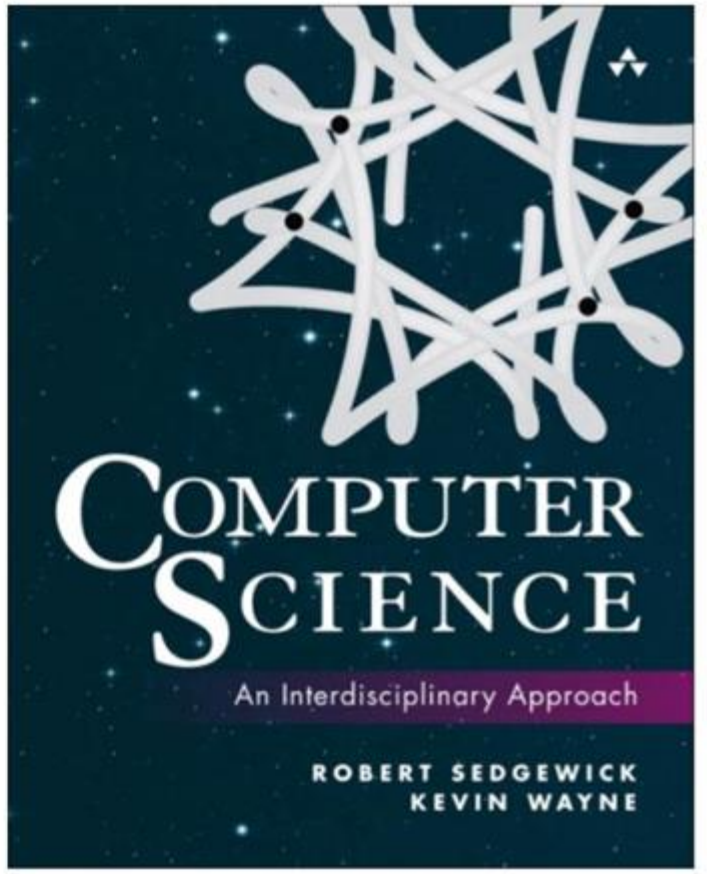
Text A

Textbook A:

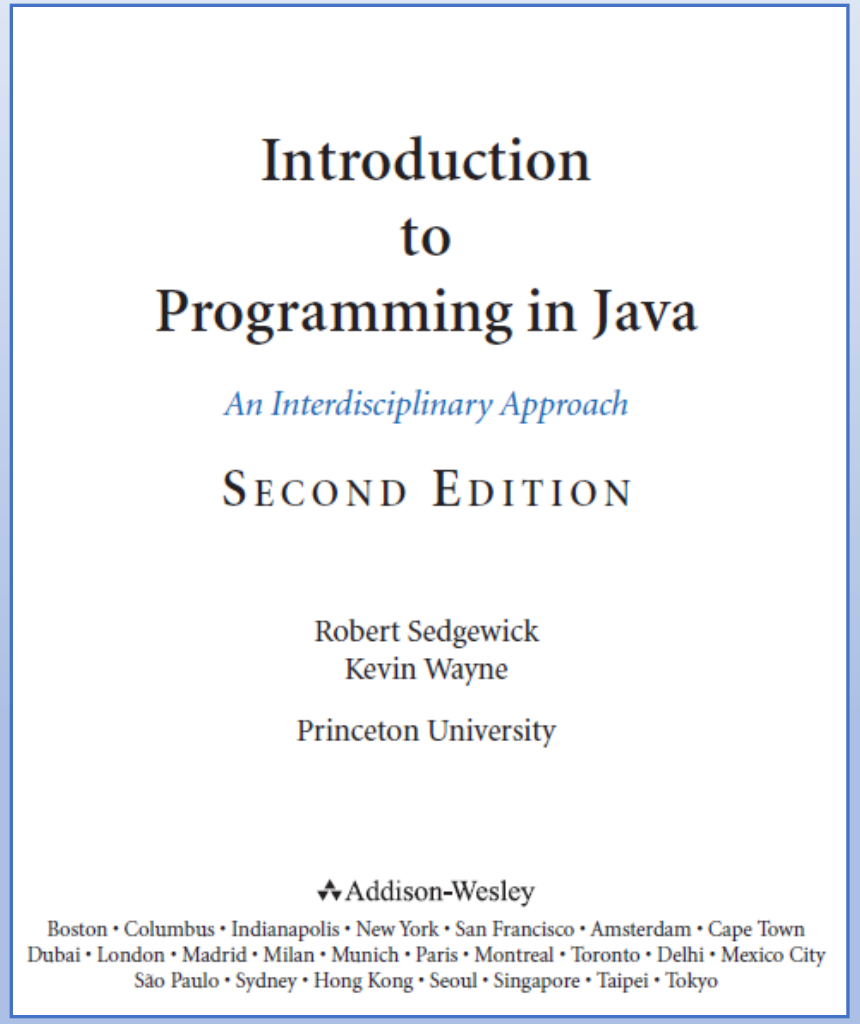
Robert Sedgewick & Kevin Wayne:

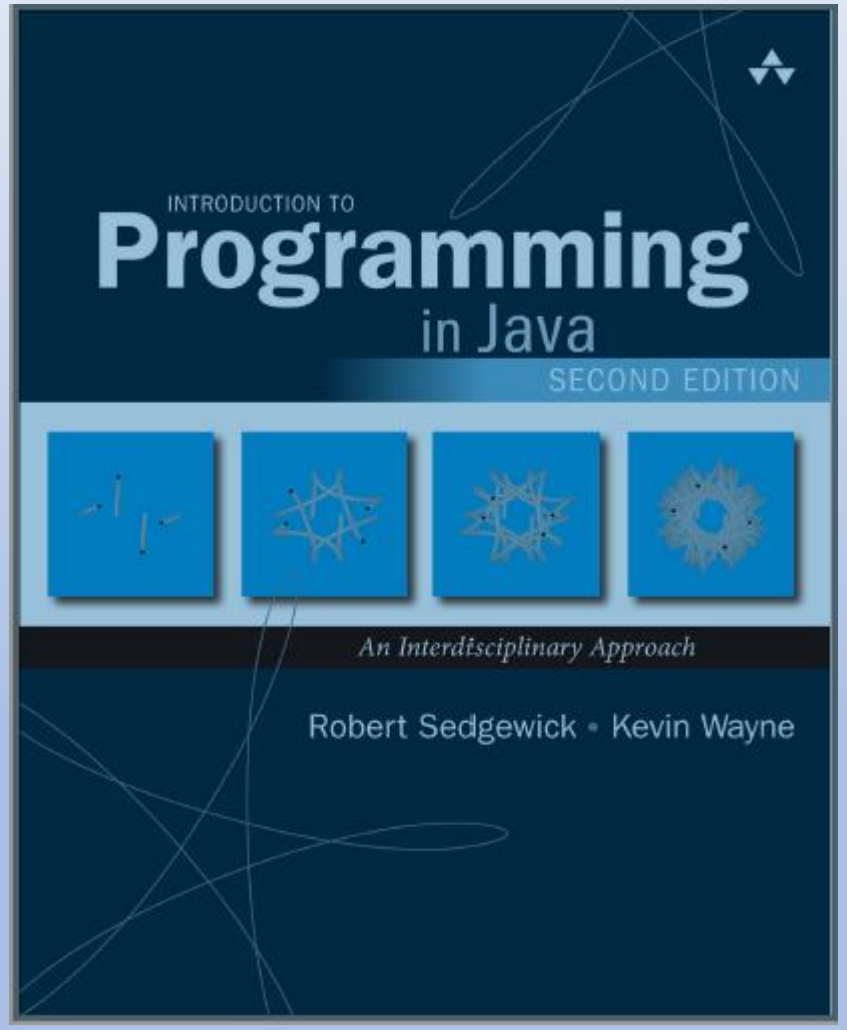
Computer Science: An Interdisciplinary Approach

2016 Addison-Wesley Professional, Pearson

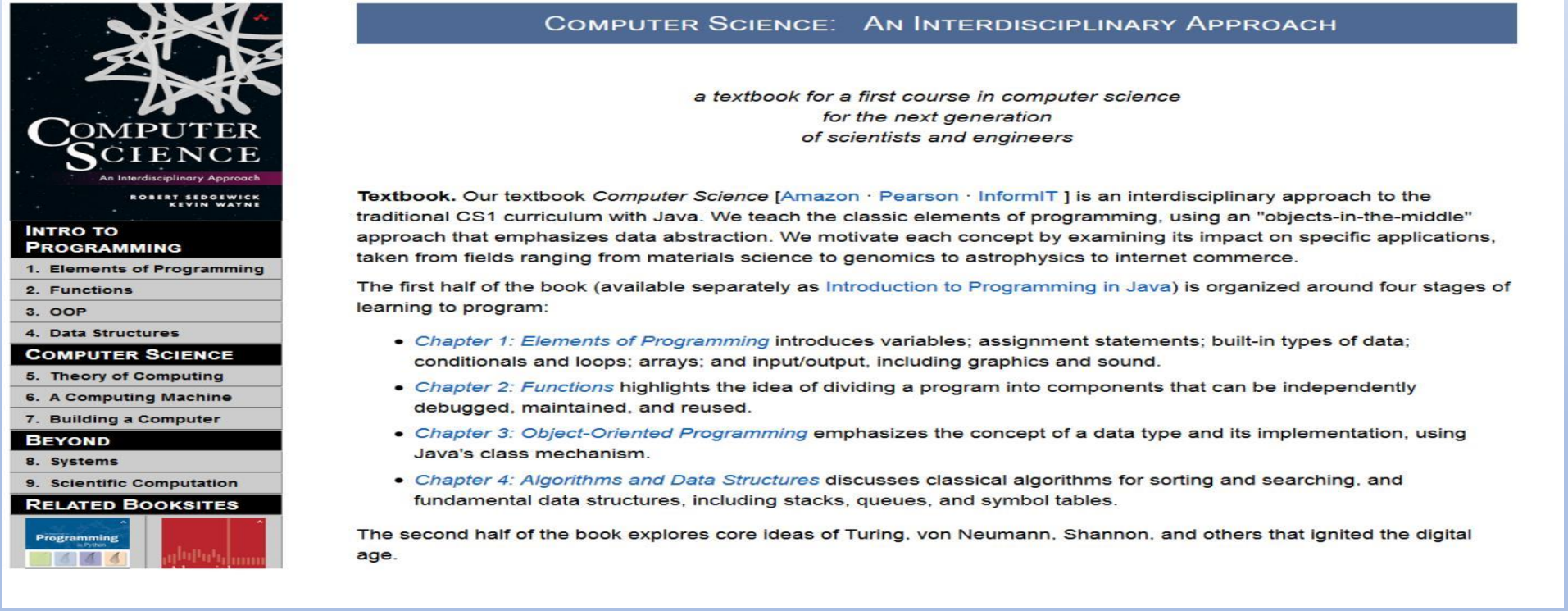


Alternative Short Version (first part of the previous book)





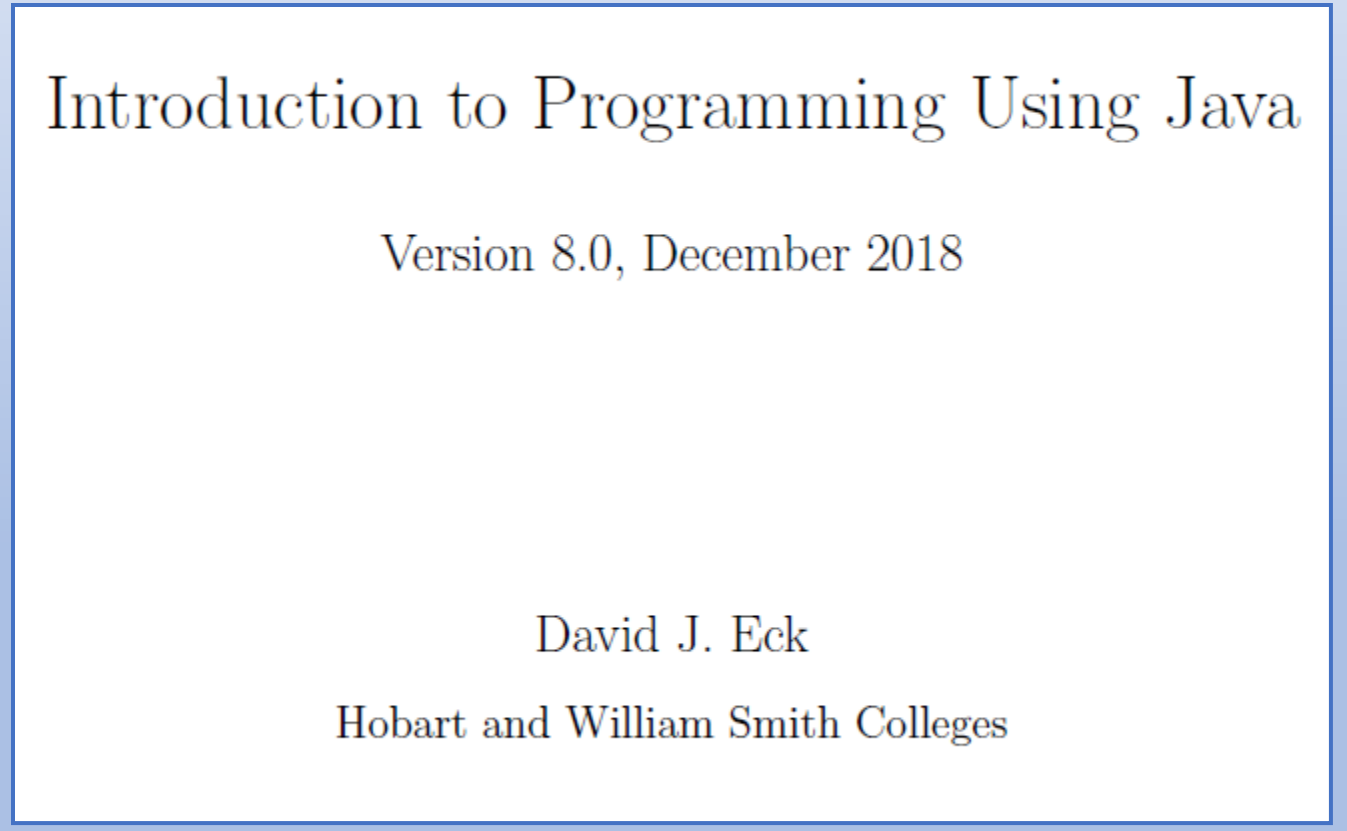
Website for the textbook
http://introcs.cs.princeton.edu/java/
You can refer to the code and slides here…



Text B

It is an online book (Thanks David J. Eck from Hobart and William Smith Colleges):

http://math.hws.edu/javanotes8/


Introduction to Programming Using Java

Version 8.0, December 2018

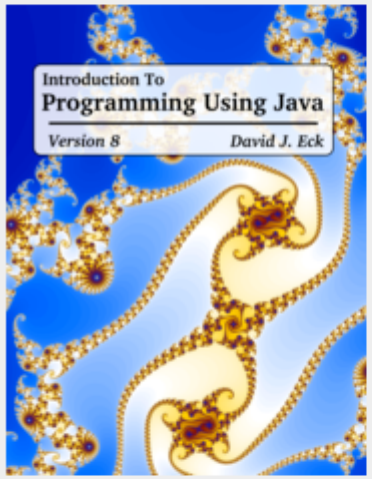
David J. Eck

Hobart and William Smith Colleges

Introduction to Programming Using Java, Eighth Edition

Version 8.0, December 2018

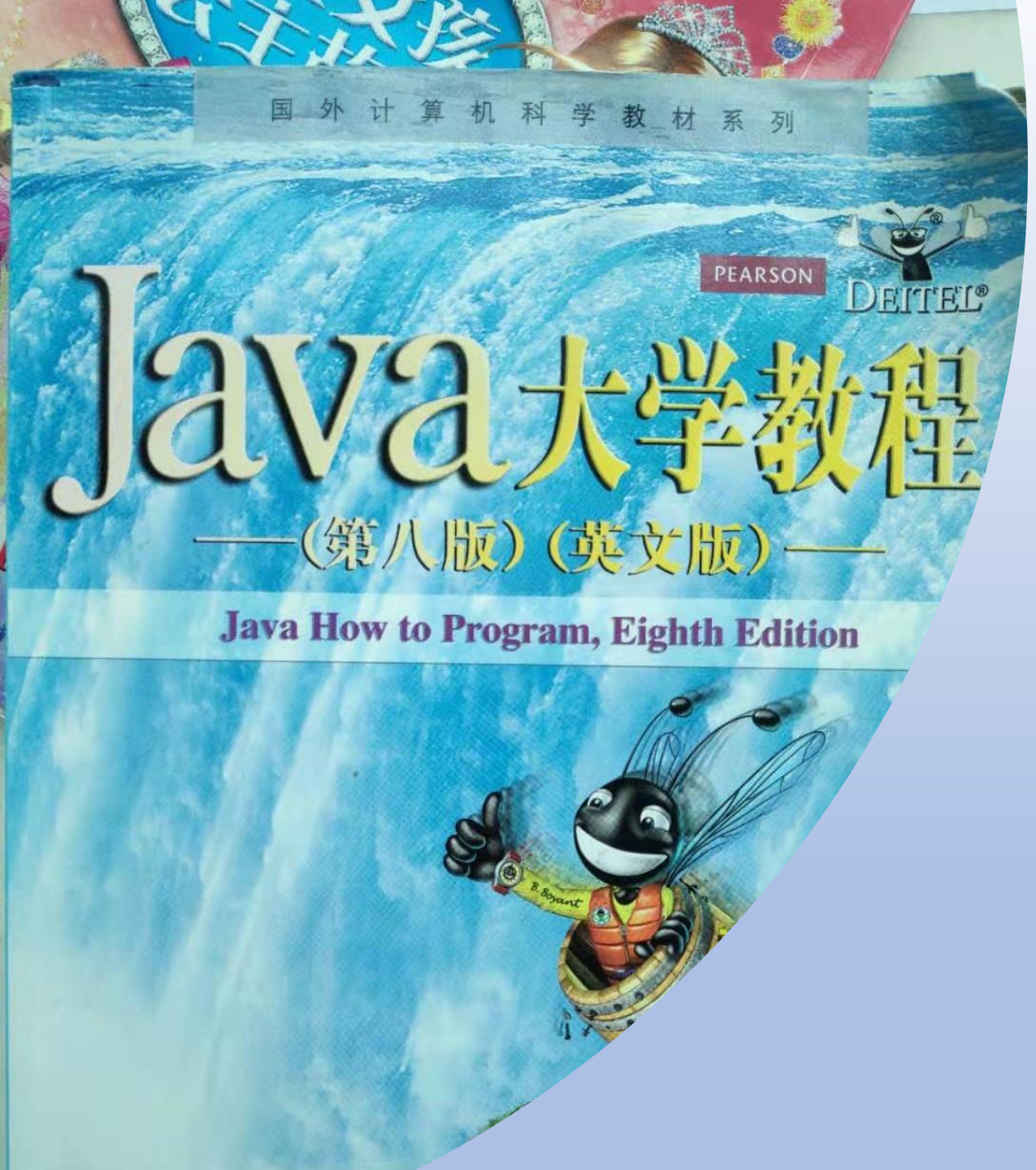
Author: David J. Eck (eck@hws.edu)



WELCOME TO the Eighth Edition of *Introduction to Programming Using Java*, a free, on-line textbook on introductory programming, which uses Java as the language of instruction. This book is directed mainly towards beginning programmers, although it might also be useful for experienced programmers who want to learn something about Java. It is certainly not meant to provide complete coverage of the Java language.

The eighth edition requires Java 8 or higher, including JavaFX. Earlier editions of the book are still available. In particular, the seventh edition uses Swing instead of JavaFX. See the preface for links to all older editions.

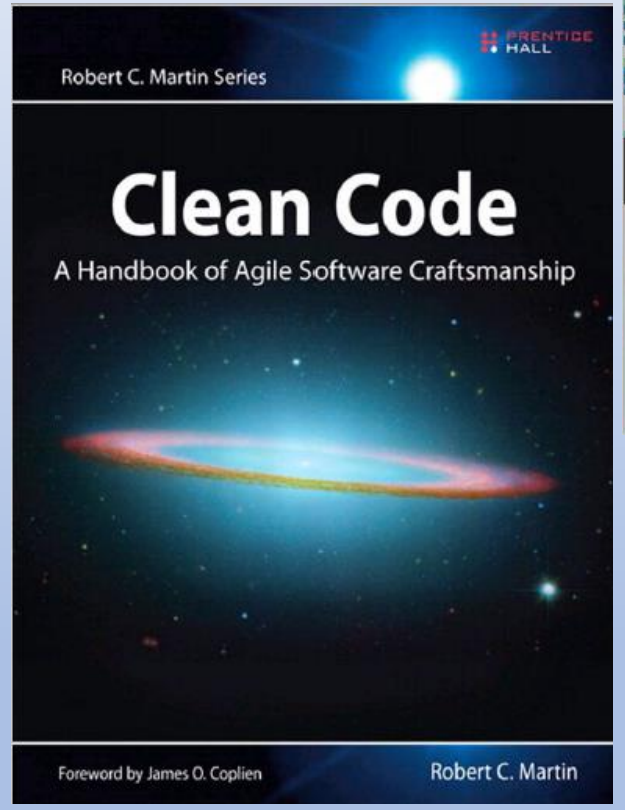
You can the download this web site for use on your own computer. PDF, e-book, and print versions of the textbook are also available. The PDF that includes links might be the best way to read it on your computer. **Links to the downloads can be found at the bottom of this page.**

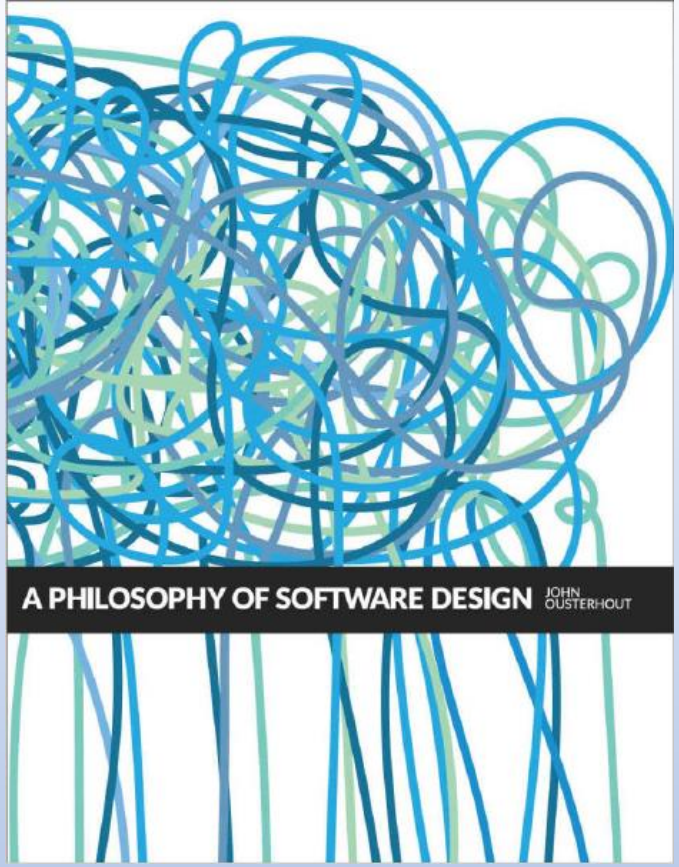


Java How to Program (any edition)

Additional reading about programming and computing to assist your development as a computing professional…

There are others that we may mention during this course…





References:

Cay Horstmann. Java Concepts, Late Objects, 3e, Wiley 2018

Robert C. Martin. Clean Code: A Handbook of Agile Software Craftsmanship. Prentice Hall, Pearson, 2009

John Ousterhout. A Philosophy of Software Design. Yaknyam Press, 2018

Important

- Most ideas will be covered in class but some details might be skipped. These details are covered in the required reading.
- Assignments and project should be done individually although group discussions are allowed
- Any dishonest behaviour and cheating in assignments will be dealt with severely
- If you get an idea for a solution from others or online you must acknowledge the source in your submission

Plagiarism Policy

* If an undergraduate assignment is found to be plagiarized, the first time the score of the assignment will be 0.

* The second time the score of the course will be 0.

As it may be difficult when two assignments are identical or nearly identical who actually wrote it, the policy will apply to BOTH students, unless one confesses having copied without the knowledge of the other.

What is okay and what is not okay?

- It's OK to work on an assignment with a friend, and think together about the program structure, share ideas and even the global logic. At the time of actually writing the code, you should write it alone.
- It's OK to use in an assignment a piece of code found on the web, as long as you indicate in a comment where it was found and don't claim it as your own work.
- It's OK to help friends debug their programs (you'll probably learn a lot yourself by doing so).
- It's OK to show your code to friends to explain the logic, as long as the friends write their code on their own later.

It's NOT OK to take the code of a friend, make a few cosmetic changes (comments, some variable names) and pass it as your own work.

What is okay and what is not okay?

南方科技大学 SOUTHERN UNIVERSITY OF SCIENCE AND TECHNOLOGY | 计算机科学与工程系 Department of Computer Science and Engineering

Undergraduate Students Assignment Declaration Form

This is_____(student ID:_____, who has enrolled in _____ course, originated the Department of Computer Science and Engineering. I have read and understood the regulations on plagiarism in assignments and theses according to "Regulations on Academic Misconduct in Assignments for Undergraduate Students in the SUSTech Department of Computer Science and Engineering". I promise that I will follow these regulations during the study of this course.

Signature:

Date:

Important message

- If you are having trouble to understand something as a question in class.

- The lecture slides and other materials are in progress. If you have a suggestion please email. Your classmates and future students will thank you.

What will I learn in this course?

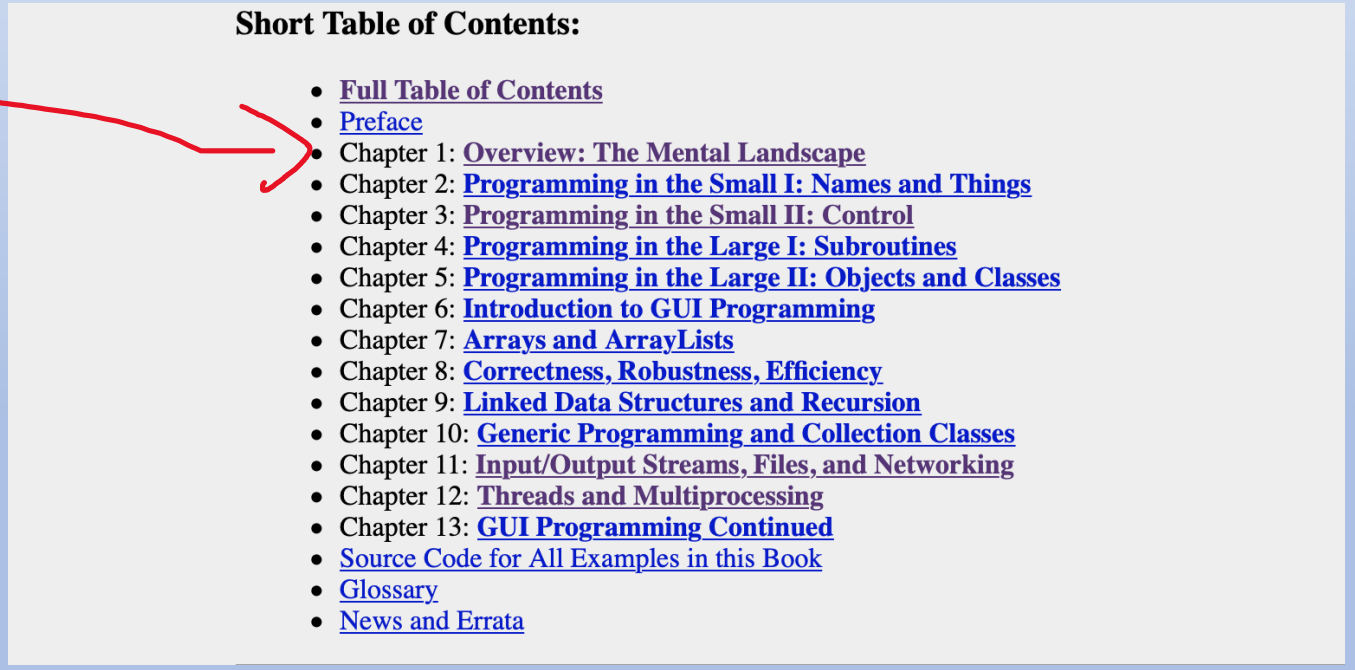
- Design principles and good software design
- Programming concepts and OO programming
- Using Java to solve real world problems efficiently
- Compared with introduction programming courses we will cover methodologies and patterns that are useful in large scale problem solving
- Some background from CS102A and CS102B is assumed

Information Systems

Information Systems

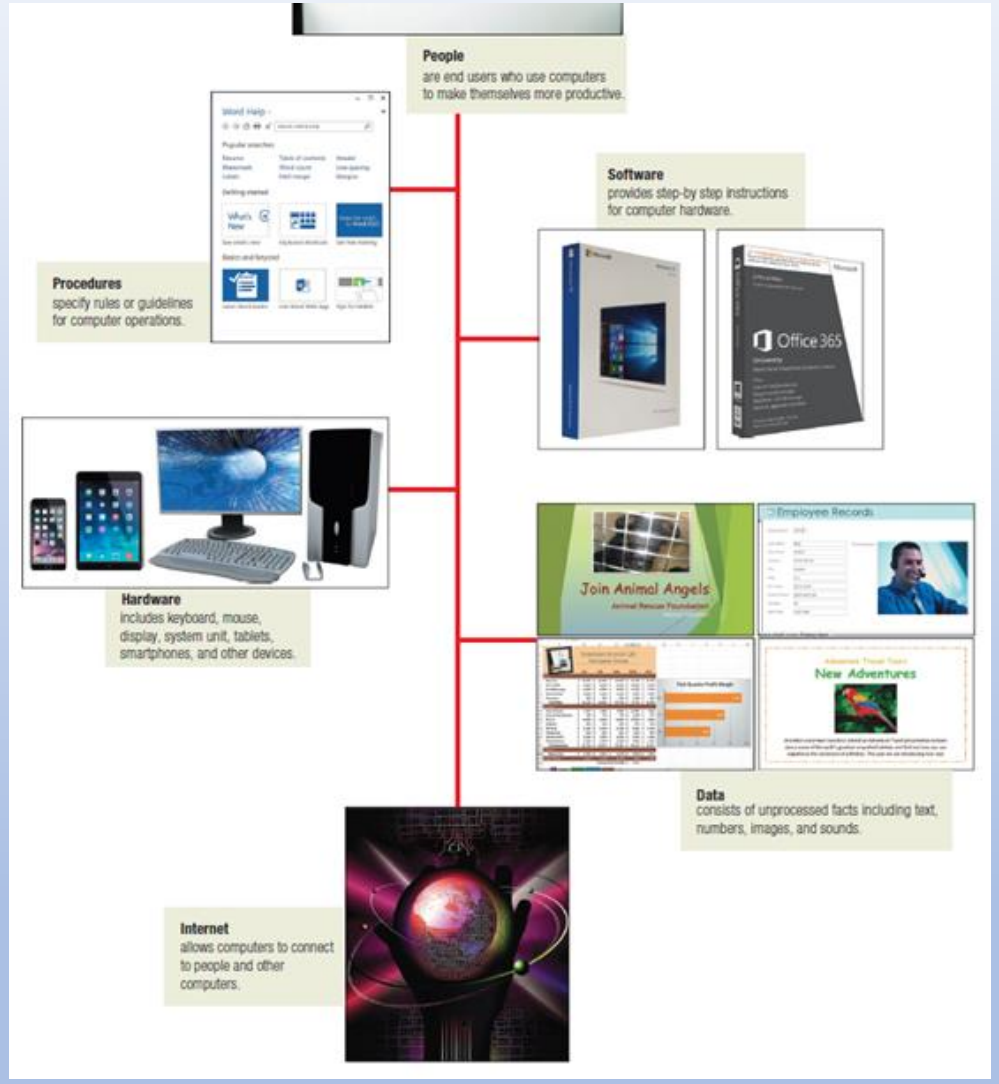
- Computing takes place in information systems

- Notes and reading:
  - Text B Chapter 1



Short Table of Contents:

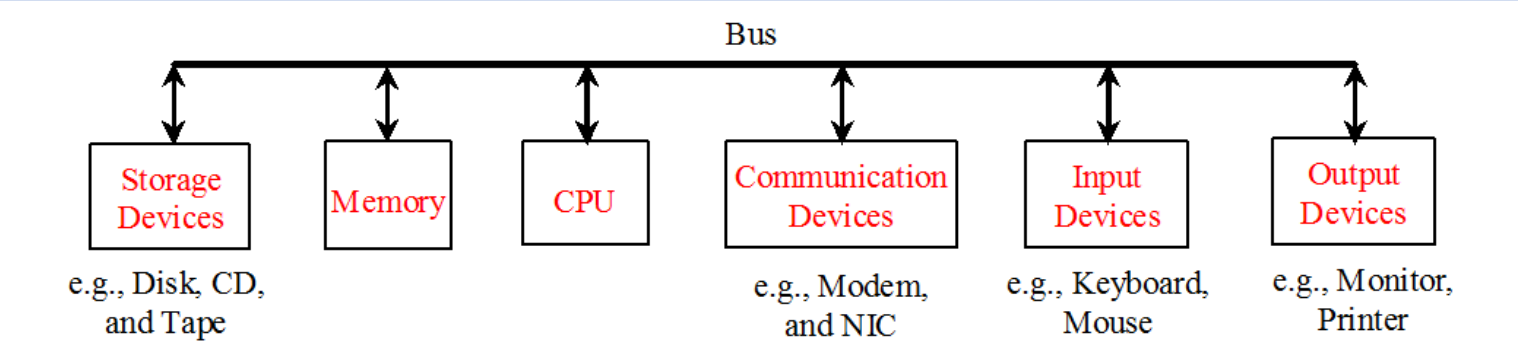
An Information System



- People
- Procedures
- Software
- Hardware
- Data
- Networks

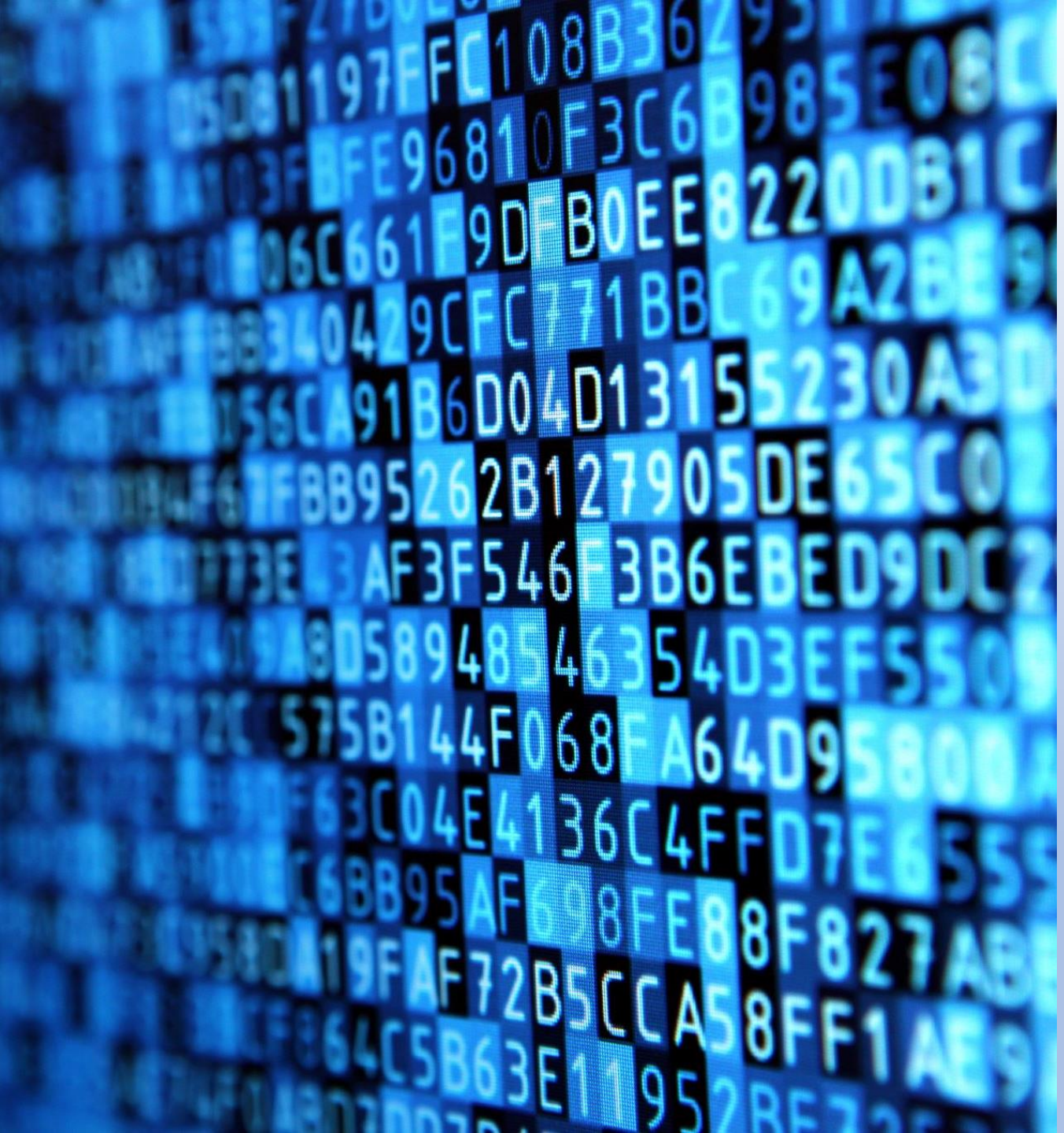
- Data VS Information

Hardware



A computer consists of various devices referred to as hardware

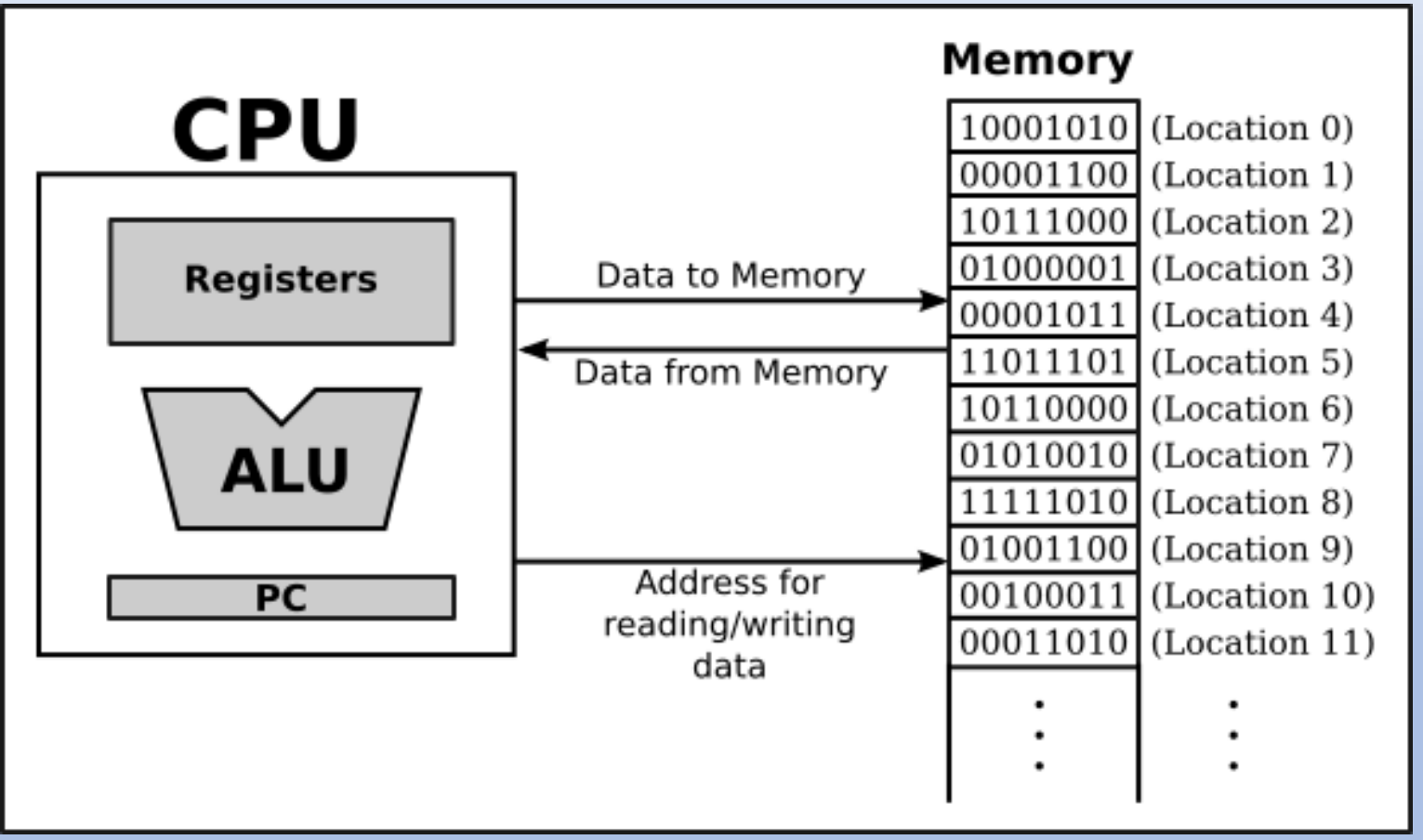
The keyboard, screen, mouse, disks, memory, DVD, CD-ROM and central processing units (CPU) are hardware



Software

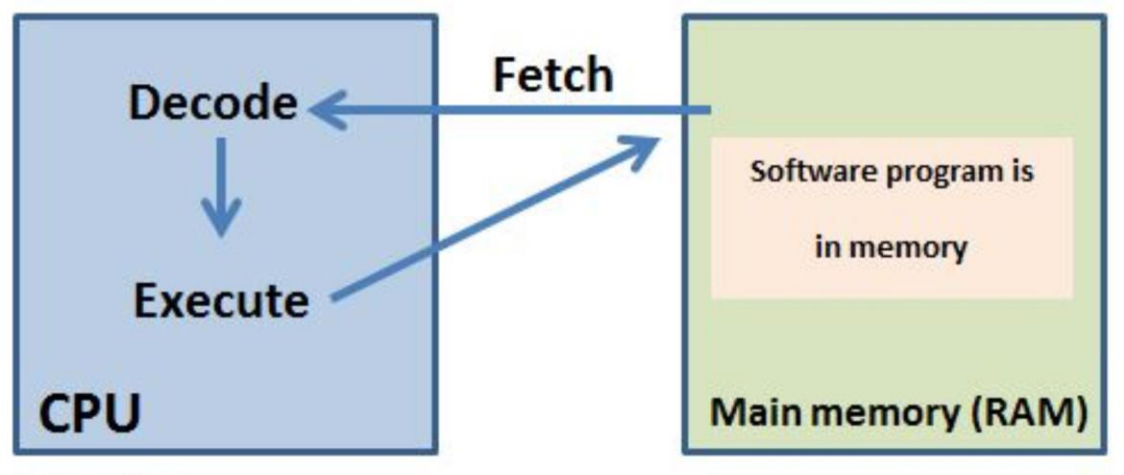
- Programs that run on a computer are referred to as software
- A program is a sequence of instructions that specify how to perform a computation
- Instructions can include statements to:
  - Input – get data
  - Output – output data
  - Math – perform calculations
  - Testing – check for conditions or run statements
  - Repetition – perform repetitive actions

Model for how computers work



- Main memory holds machine language programs and data encoded as binary numbers
- The CPU fetches instructions from memory in sequence and executes them
- Each instruction is a very small task (e.g. adding 2 numbers)
- The program must be complete and unambiguous as the CPU executes everything exactly as written

Fetch Execute Cycle

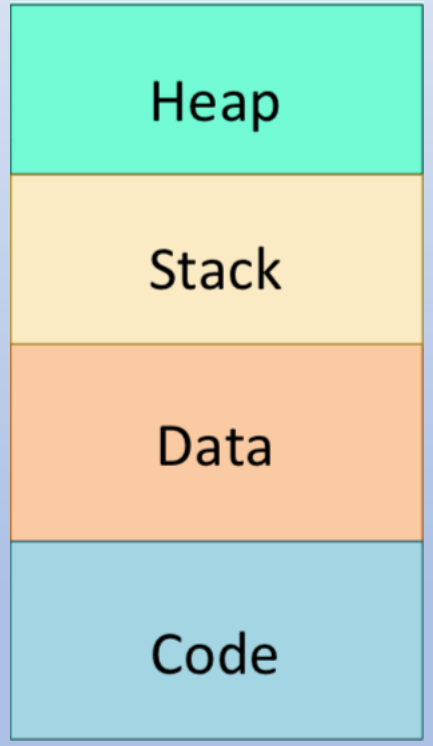


- **The Fetch Execute Cycle** is the basic cycle of how computers operate to process instructions
- During the cycle the computer receives a program instruction from memory, then it establishes and carries out the actions specified
- **Fetch** – gets the next program command from the computer's memory
- **Decode** – deciphers what the program is telling the computer to do
- **Execute** – carries out the requested action
- **Store** – saves the results to a Register or Memory

John von Neumann

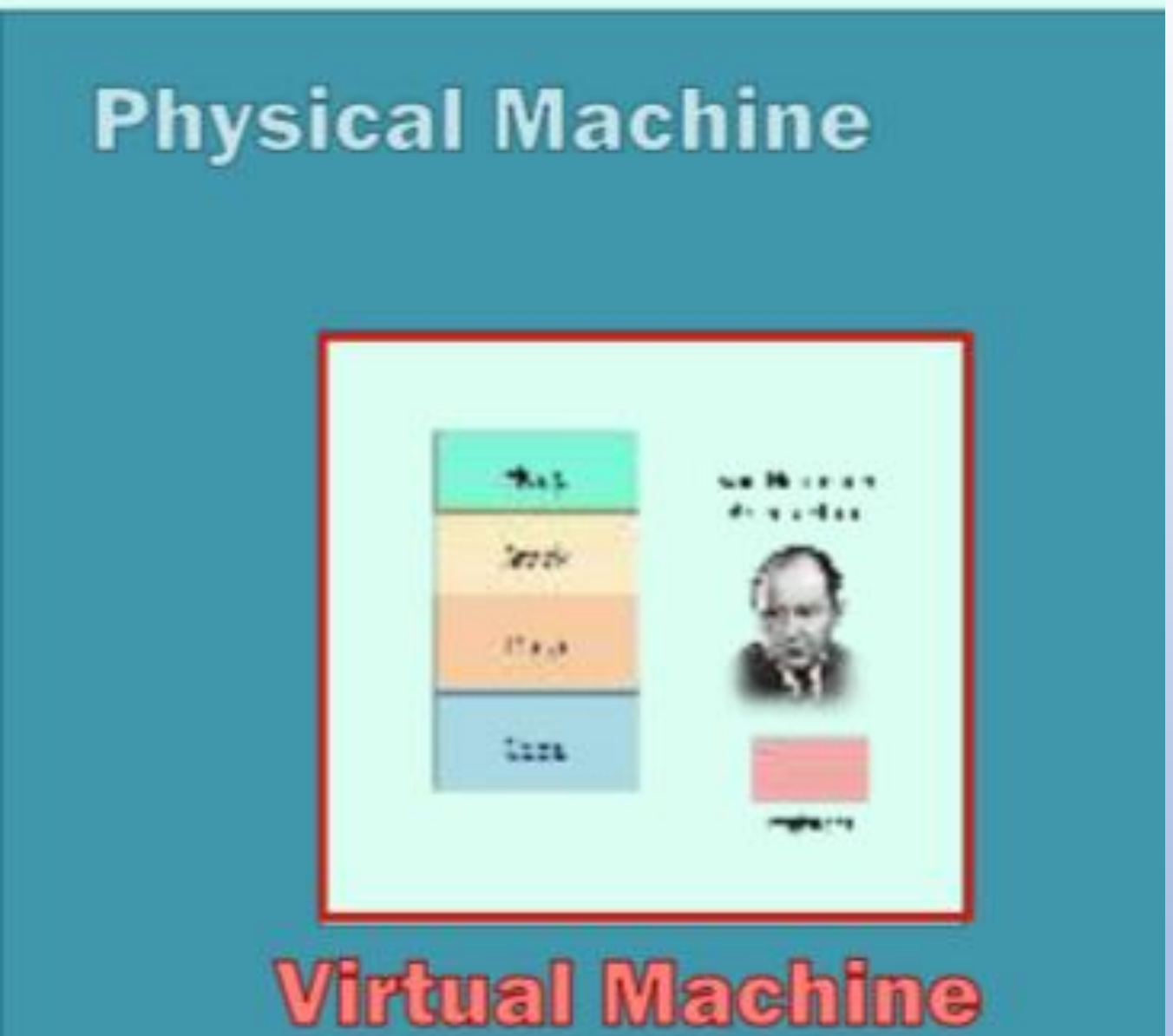
- The **fetch execute cycle** was first proposed by John von Neumann who is famous for the Von Neumann architecture, the framework which is being followed by most computers today.

Von Neumann Architecture



You are probably aware of the "von Neumann Architecture" that was defined (not invented) by John von Neuman and in which the computer memory is split between a part that contains executable instructions (code), a part that contains static data, a stack where volatile data associated with functions come and go, and a heap where objects are created. The processor also contains registers where data and instructions are brought from memory for processing.

JAVA Virtual Machine

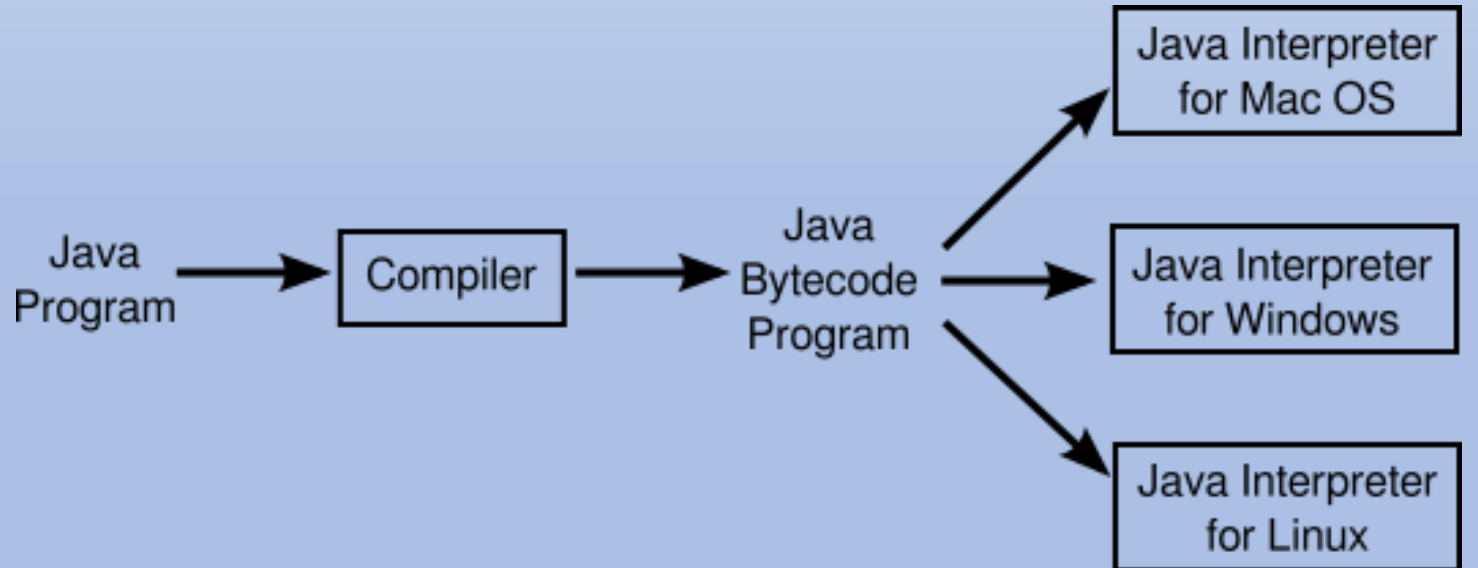


The Java Virtual Machine simulates a Von Neuman machine in a standard way on a lot of physical machines. Implementations of JVMs on different machines are different, but what they run is the same.

It is often said “With java – write once – run everywhere”

- The CPU executes instructions written in Machine Code (a very simple set of instructions that can be executed directly by the CPU)
- Usually programs are written in high-level programming languages such as Java, Python, or C++. A program written in a high-level language cannot be run directly on any computer and has to be translated.
- This translation can be done by a program called a compiler.
- Once the translation is done, the machine-language program can be run any number of times, but of course it can only be run on one type of computer (since each type of computer has its own individual machine language).
- Instead of using a compiler, we can use an **interpreter**, which translates each instruction, as necessary. An interpreter is a program that acts much like a CPU, with a kind of fetch-and-execute cycle. In order to execute a program, the interpreter runs in a loop in which it repeatedly reads one instruction from the program, decides what is necessary to carry out that instruction, and then performs the appropriate machine-language commands to do so.

Java Virtual Machine

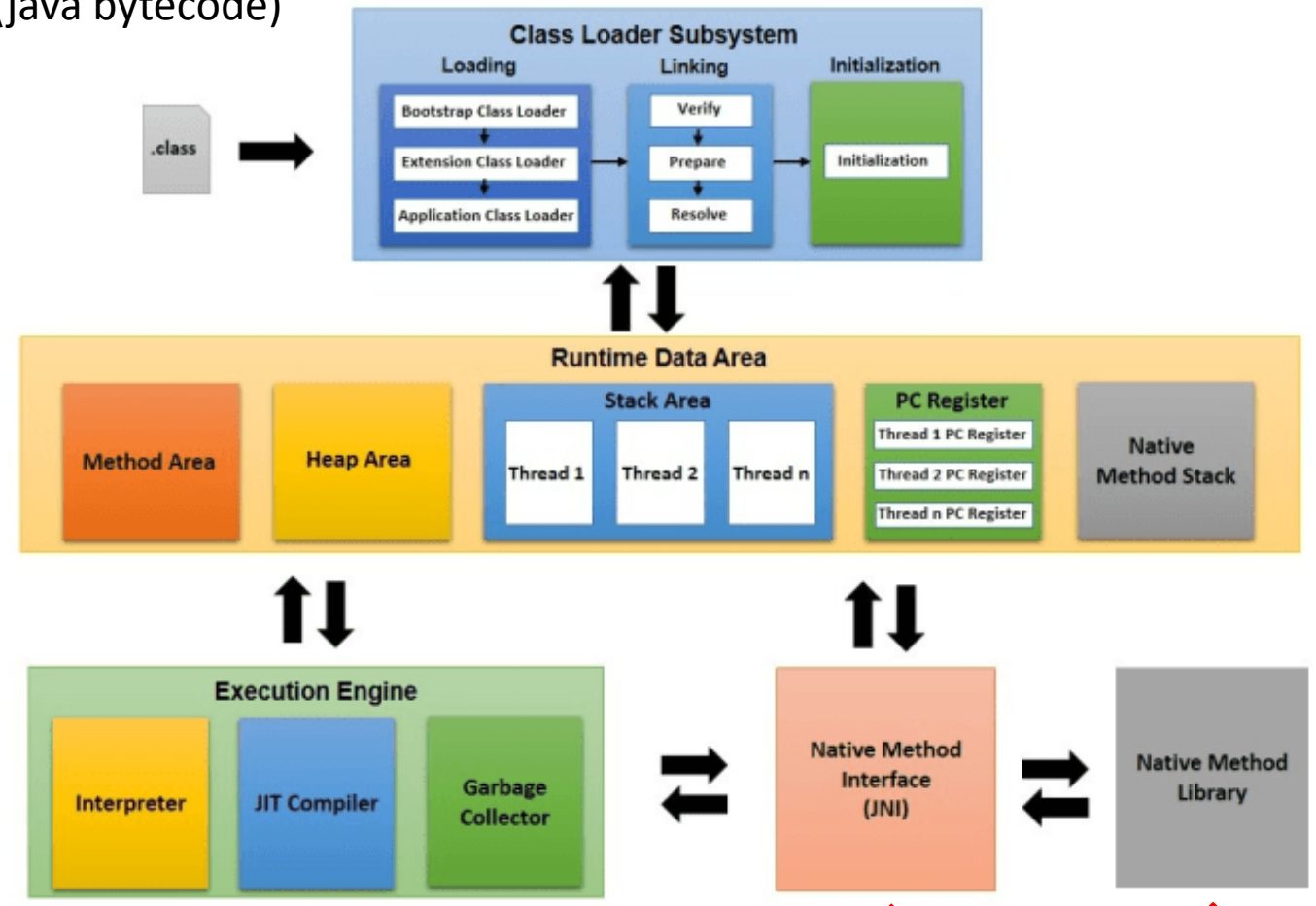


Other Examples of Virtualization

- In essence the JVM is a virtual representation of a type of ideal machine
- Virtualization hides the physical characteristics of a computing platform from the users, presenting instead an abstract computing platform
- There are other approaches to virtualization such as:
  - Simulating a hardware system in software and installing any operating system on it (eg vmware)
  - Simulating another different operating system on machine (eg Docker and containers)

How the Java Virtual Machine Works

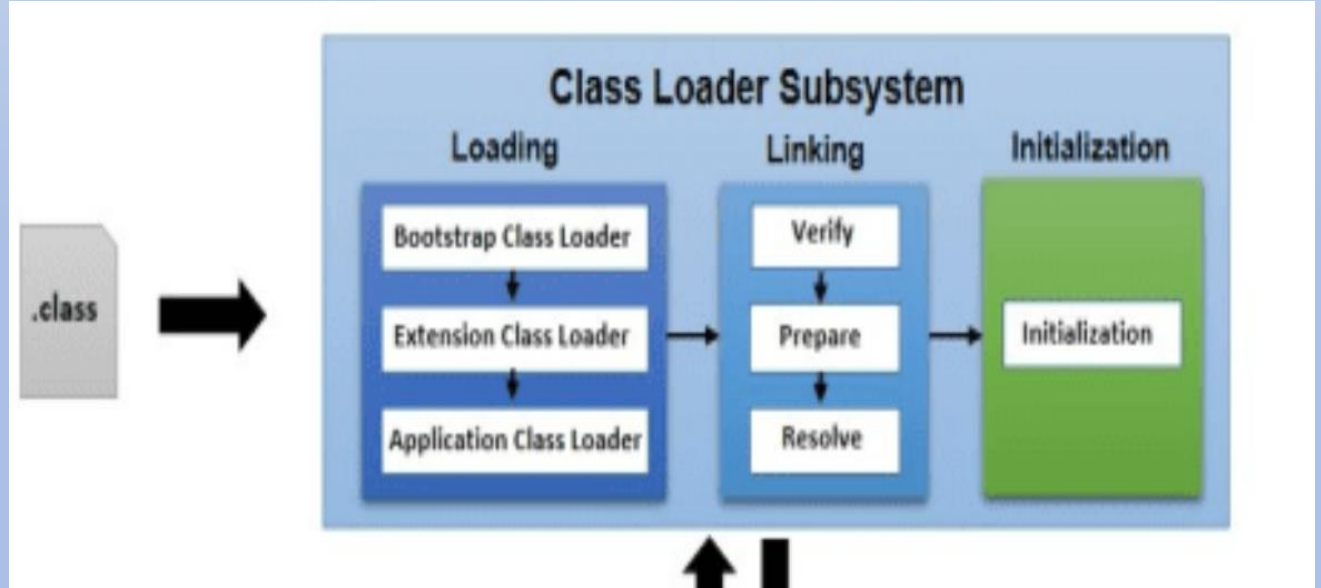
Class file
(java bytecode)



A JVM can be divided into three main components, Class Loader, Runtime Data Areas and Execution Engine. "Native" in the schema refers implementation on a particular system.

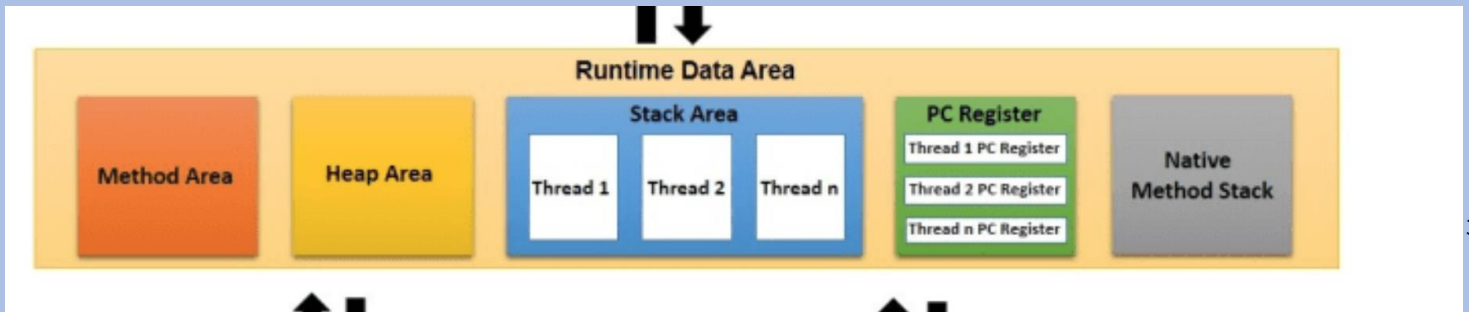
The class loader is in charge of loading the .class files (sometimes grouped together into a .jar**, J**ava **AR**chive). Those files contain intermediate code which is no longer Java code but not yet computer instructions. One of the tasks of the class loader is to locate every single class referenced in the code (everything that was imported, but also the classes that are referenced but not defined in your program, and for which a .class is expected to be found in the CLASSPATH, a list of directories or .jar files), then to set up everything so that when you call a method the engine knows where to find the instructions to execute. Finally it will initialize everything and you'll be able to call main().

Class Loader Subsystem



Runtime Data Areas

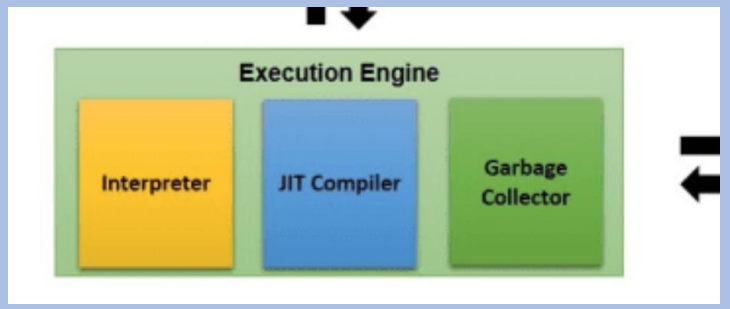
- You will find in the Runtime Data Areas all the components from the von Neumann architecture, with built-in support for threads, which are tasks performed independently and concurrently (at the same time).
- A single machine can have many thousands of threads!



Execution Engine

The execution engines contains an interpreter that converts the "run everywhere" code found in the .class to some "run on this particular hardware" instructions that the processor can execute. Interpreting code is slow (especially when running loops and repeating instructions), and pretty soon was introduced in Java a "Just In Time Compiler" that converts but then reuses the conversion when the same code is executed again.

As the JIT compiler discovers the code as it executes, every optimization cannot be executed from the start, and there is a sophisticated mechanism. The last component is the garbage collector that identifies objects that are no longer in use and reclaims the memory they are using.



Java and programming basics

Basics of programming (note: this should be revision)

Built in types

Conditionals

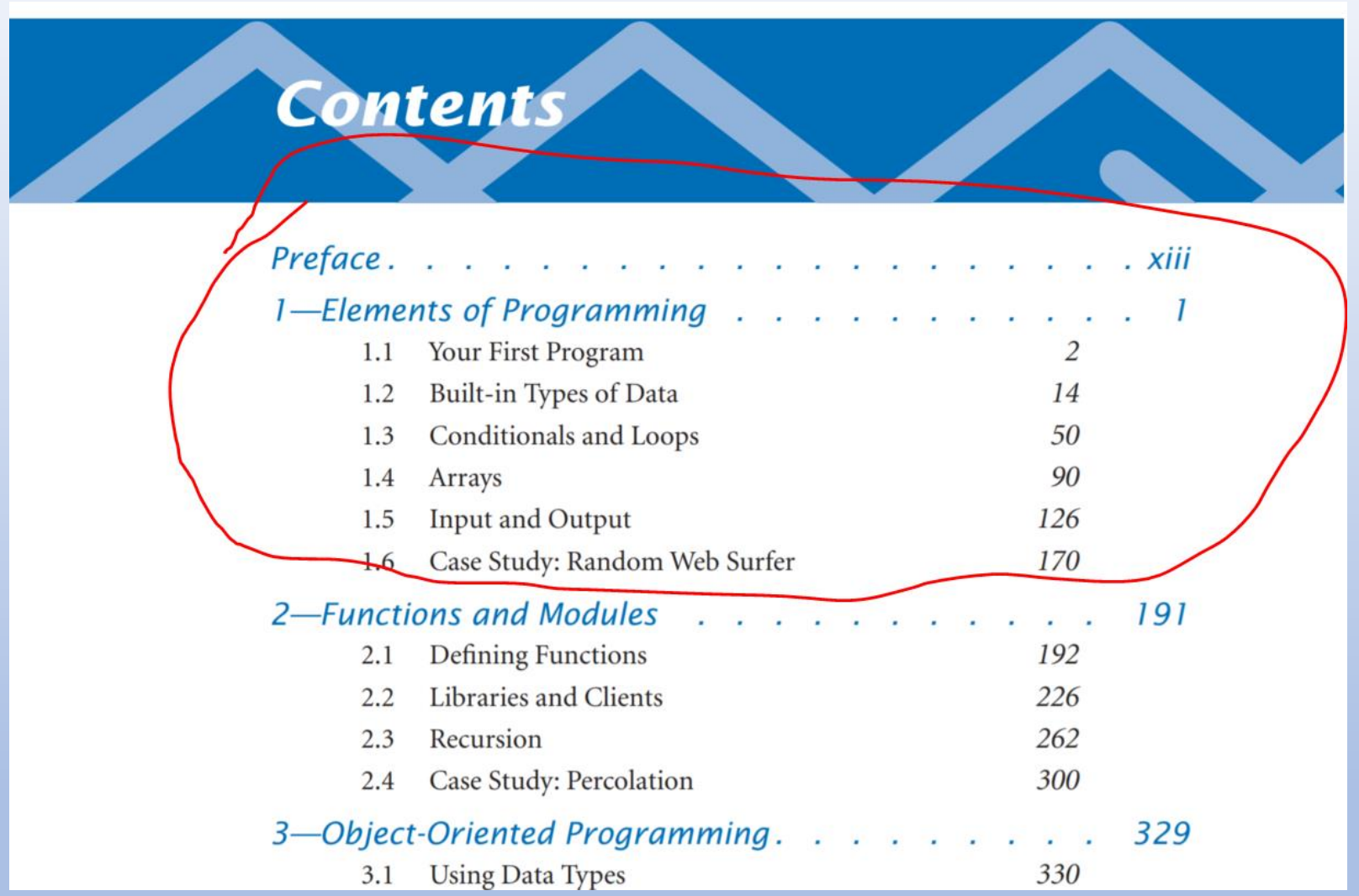
Loops

Arrays

I/O

Review

See textbook Sedgwick Ch 1 for a complete review…



Contents

Example of inbuilt type double

Compilation: javac Quadratic.java

Execution: java Quadatic b c

Given b and c, solves for the roots of x*x + b*x + c.

Quadratic formula (the implementation assumes a = 1)

$$x = \frac{-b \pm \sqrt{b^2 - 4ac}}{2a}$$

```
public class Quadratic {

    public static void main(String[] args) {
        double b = Double.parseDouble(args[0]);
        double c = Double.parseDouble(args[1]);

        double discriminant = b*b - 4.0*c;
        double sqroot =  Math.sqrt(discriminant);

        double root1 = (-b + sqroot) / 2.0;
        double root2 = (-b - sqroot) / 2.0;

        System.out.println(root1);
        System.out.println(root2);
    }
}
```

Loops

Simulation: gamblers ruin

Suppose that a gambler makes a series of fair $1 bets, starting with some given initial stake. The gambler always goes broke eventually, but when we set other limits on the game, various questions arise.

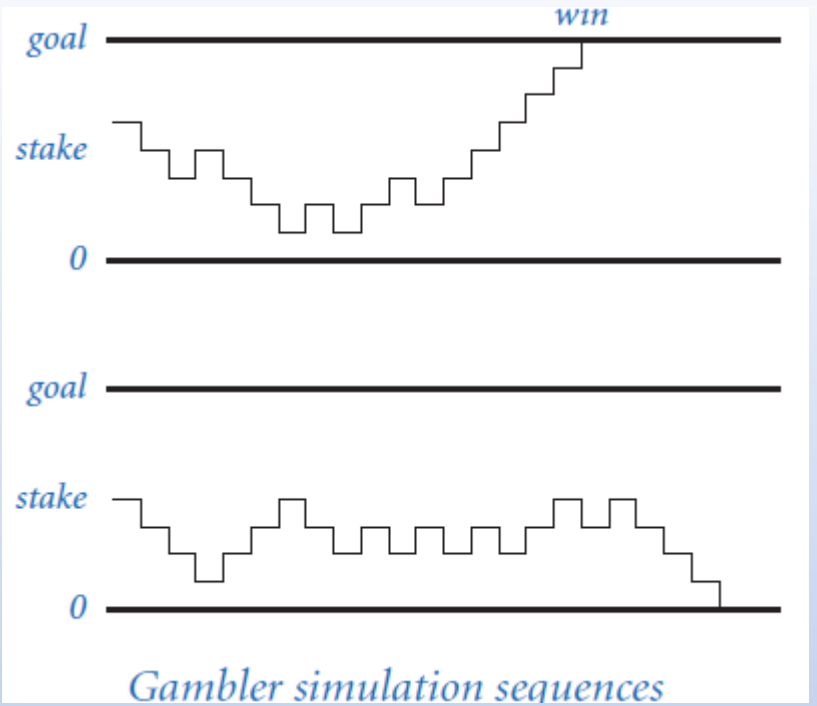
For example, suppose that the gambler decides ahead of time to walk away after reaching a certain goal. What are the chances that the gambler will win? How many bets might be needed to win or lose the game? What is the maximum amount of money that the gambler will have during the course of the game?

probability of success is the ratio of the stake to the goal and that the *expected number of bets is the product of the stake and the desired gain* (the difference between the goal and the stake). For example, if you go to Monte Carlo to try to turn $500 into $2,500, you have a reasonable (20%) chance of success, but you should expect to make a million $1 bets! If you try to turn $1 into $1,000, you have a 0.1% chance and can expect to be done (ruined, most likely) in about 999 bets.

```
public class Gambler
{
   public static void main(String[] args)
   {  // Run trials experiments that start with
      // $stake and terminate on $0 or $goal.
      int stake  = Integer.parseInt(args[0]);
      int goal   = Integer.parseInt(args[1]);
      int trials = Integer.parseInt(args[2]);
      int bets = 0;
      int wins = 0;
      for (int t = 0; t < trials; t++)
      {  // Run one experiment.
         int cash = stake;
         while (cash > 0 && cash < goal)
         {  // Simulate one bet.
            bets++;
            if (Math.random() < 0.5) cash++;
            else                     cash--;
         }  // Cash is either 0 (ruin) or $goal (win).
         if (cash == goal) wins++;
      }
      System.out.println(100*wins/trials + "% wins");
      System.out.println("Avg # bets: " + bets/trials);
   }
}
```

stake	*initial stake*
goal	*walkaway goal*
trials	*number of trials*
bets	*bet count*
wins	*win count*
cash	*cash on hand*

Runtime = trials * bets



Gambler simulation sequences

```
% java Gambler 50 250 100
19% wins
Avg # bets: 11050
% java Gambler 500 2500 100
21% wins
Avg # bets: 998071
```

```
% java Gambler 10 20 1000
50% wins
Avg # bets: 100
% java Gambler 10 20 1000
51% wins
Avg # bets: 98
```

Finding factors

Factoring. A prime number is an integer greater than 1 whose only positive divisors are 1 and itself.

The prime factorization of an integer is the multiset of primes whose product is the integer. For example, 3,757,208 = 2 x 2 x 2 x 7 x 13 x 13 = 397.

Here is a class Factors to compute the prime factorization of any given positive integer.

```
% java Factors 3757208
2 2 2 7 13 13 397
```

```
public class Factors {

   public static void main(String[] args) {

      // command-line argument
      long n = Long.parseLong(args[0]);

      System.out.print("The prime factorization of " + n + " is: ");

      // for each potential factor
      for (long factor = 2; factor*factor <= n; factor++) {

         // if factor is a factor of n, repeatedly divide it out
         while (n % factor == 0) {
            System.out.print(factor + " ");
            n = n / factor;
         }
      }

      // if biggest factor occurs only once, n > 1
      if (n > 1) System.out.println(n);
      else       System.out.println();
   }
}
```

```
* % java Factors 81
* The prime factorization of 81 is: 3 3 3 3
*
* % java Factors 168
* The prime factorization of 168 is: 2 2 2 3 7
```

factor	*n*	*output*
2	3757208	2 2 2
3	469651	
4	469651	
5	469651	
6	469651	
7	469651	7
8	67093	
9	67093	
10	67093	
11	67093	
12	67093	
13	67093	13 13
14	397	
15	397	
16	397	
17	397	
18	397	
19	397	
20	397	
		397

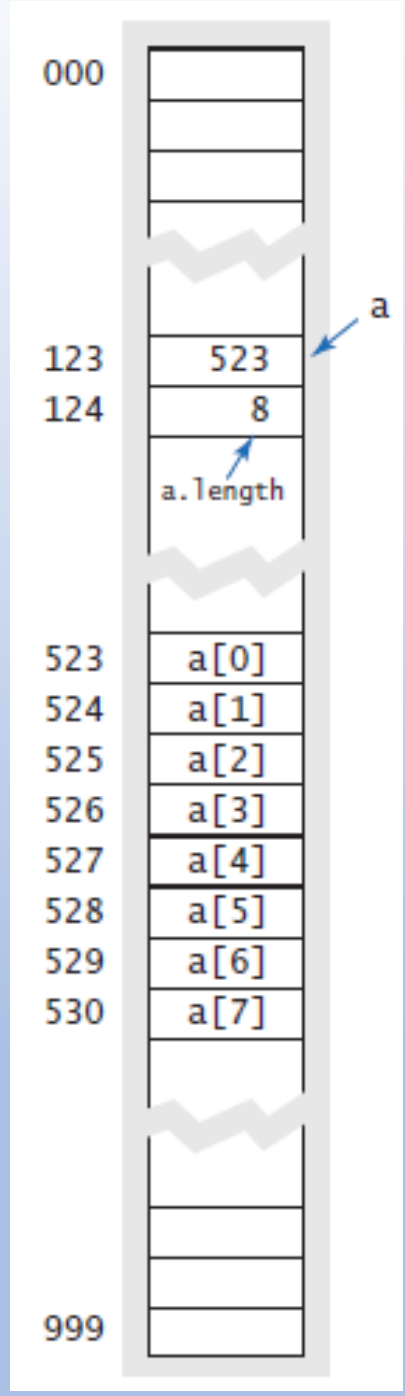
Trace of **java Factors** *3757208*

Arrays

Memory representation

int[] a = new int [7];

When you use the keyword new to create an array, Java reserves sufficient space in memory to store the specified number of elements. This process is called *memory allocation*. The same process is required for all variables that you use in a program (but you do not use the keyword new with variables of primitive types because Java knows how much memory to allocate). It is your responsibility to create an array before accessing any of its elements. If you fail to adhere to this rule, you will get an *uninitialized variable* error at compile time.



Simulation

Arrays

Example: coupon collector

r	isCollected[] 0	1	2	3	4	5	distinct	count
	F	F	F	F	F	F	0	0
2	F	F	T	F	F	F	1	1
0	T	F	T	F	F	F	2	2
4	T	F	T	F	T	F	3	3
0	T	F	T	F	T	F	3	4
1	T	T	T	F	T	F	4	5
2	T	T	T	F	T	F	4	6
5	T	T	T	F	T	T	5	7
0	T	T	T	F	T	T	5	8
1	T	T	T	F	T	T	5	9
3	T	T	T	T	T	T	6	10

Trace for a typical run of `java CouponCollector 6`

"How many cards do you need to turn up before you have seen one of each suit?"

```java
public class CouponCollector {
    public static void main(String[] args) {
        int n = Integer.parseInt(args[0]);        // number of card types
        boolean[] isCollected = new boolean[n];  // isCollected[i] = true if card i has been collected
        int count = 0;                            // total number of cards collected
        int distinct = 0;                         // number of distinct cards

        // repeatedly choose a random card and check whether it's a new one
        while (distinct < n) {
            int value = (int) (Math.random() * n);   // random card between 0 and n-1
            count++;                                 // we collected one more card
            if (!isCollected[value]) {
                distinct++;
                isCollected[value] = true;
            }
        }

        // print the total number of cards collected
        System.out.println(count);
    }
}
```

Sieve of Eratosthenes

An algorithm for making tables of primes. Sequentially write down the integers from 2 to the highest number n you wish to include in the table. Cross out all numbers >2 which are divisible by 2 (every second number). Find the smallest remaining number >2. It is 3. So cross out all numbers >3 which are divisible by 3 (every third number). Find the smallest remaining number >3. It is 5. So cross out all numbers >5 which are divisible by 5 (every fifth number).

The sieve of Eratosthenes can be used to compute the prime counting function.

The prime counting function $\pi(n)$ is the number of primes less than or equal to n. For example $\pi(17) = 7$ since the first seven primes are 2, 3, 5, 7, 11, 13, and 17.

- https://mathworld.wolfram.com/PrimeCountingFunction.html
- https://mathworld.wolfram.com/SieveofEratosthenes.html

Sieve of Eratosthenes

n	Primes <= n
10	4
100	25
1,000	168
10,000	1,229
100,000	9,592
1,000,000	78,498
10,000,000	664,579
100,000,000	5,761,455
1,000,000,000	50,847,534

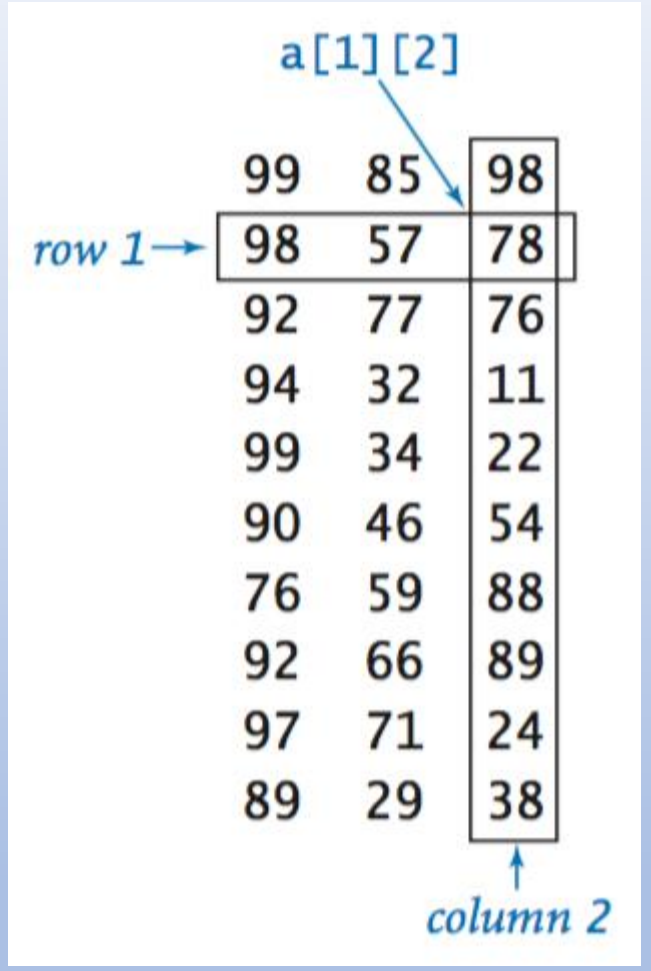
```java
public class PrimeSieve {
    public static void main(String[] args) {
        int n = Integer.parseInt(args[0]);

        // initially assume all integers are prime
        boolean[] isPrime = new boolean[n+1];
        for (int i = 2; i <= n; i++) {
            isPrime[i] = true;
        }

        // mark non-primes <= n using Sieve of Eratosthenes
        for (int factor = 2; factor*factor <= n; factor++) {

            // if factor is prime, then mark multiples of factor as nonprime
            // suffices to consider mutiples factor, factor+1, ...,  n/factor
            if (isPrime[factor]) {
                for (int j = factor; factor*j <= n; j++) {
                    isPrime[factor*j] = false;
                }
            }
        }

        // count primes
        int primes = 0;
        for (int i = 2; i <= n; i++) {
            if (isPrime[i]) primes++;
        }
        System.out.println("The number of primes <= " + n + " is " + primes);
    }
}
```

Arrays

2 dimensions

```
double[][] a = new double[m][n];
```

Two-dimensional arrays in Java. To refer to the element in row `i` and column `j` of a two-dimensional array `a[][]`, we use the notation `a[i][j]`; to declare a two-dimensional array, we add another pair of brackets; to create the array, we specify the number of rows followed by the number of columns after the type name...



Initialize an array

```
double[][] a;
a = new double[m][n];
for (int i = 0; i < m; i++)
   for (int j = 0; j < n; j++)
      a[i][j] = 0;
```

Memory representation. Java represents a two-dimensional array as an array of arrays. A matrix with `m` rows and `n` columns is actually an array of length `m`, each entry of which is an array of length `n`. In a two-dimensional Java array, we can use the code `a[i]` to refer to the ith row (which is a one-dimensional array). Enables ragged arrays.

Setting values at compile time.

To initialize an 11-by-4 array a[][]:

```
double[][] a = {
   { 99.0, 85.0, 98.0, 0.0 },
   { 98.0, 57.0, 79.0, 0.0 },
   { 92.0, 77.0, 74.0, 0.0 },
   { 94.0, 62.0, 81.0, 0.0 },
   { 99.0, 94.0, 92.0, 0.0 },
   { 80.0, 76.5, 67.0, 0.0 },
   { 76.0, 58.5, 90.5, 0.0 },
   { 92.0, 66.0, 91.0, 0.0 },
   { 97.0, 70.5, 66.5, 0.0 },
   { 89.0, 89.5, 81.0, 0.0 },
   {  0.0,  0.0,  0.0, 0.0 }
};
```

Ragged arrays

```
for (int i = 0; i < a.length; i++) {
    for (int j = 0; j < a[i].length; j++) {
        System.out.print(a[i][j] + " ");
    }
    System.out.println();
}
```

There is no requirement that all rows in a two-dimensional array have the same length—an array with rows of nonuniform length is known as a *ragged array*. The possibility of ragged arrays creates the need for more care in crafting array-processing code.

Multidimensional arrays

```
double[][][] a = new double[n][n][n];
```

3 dimensions

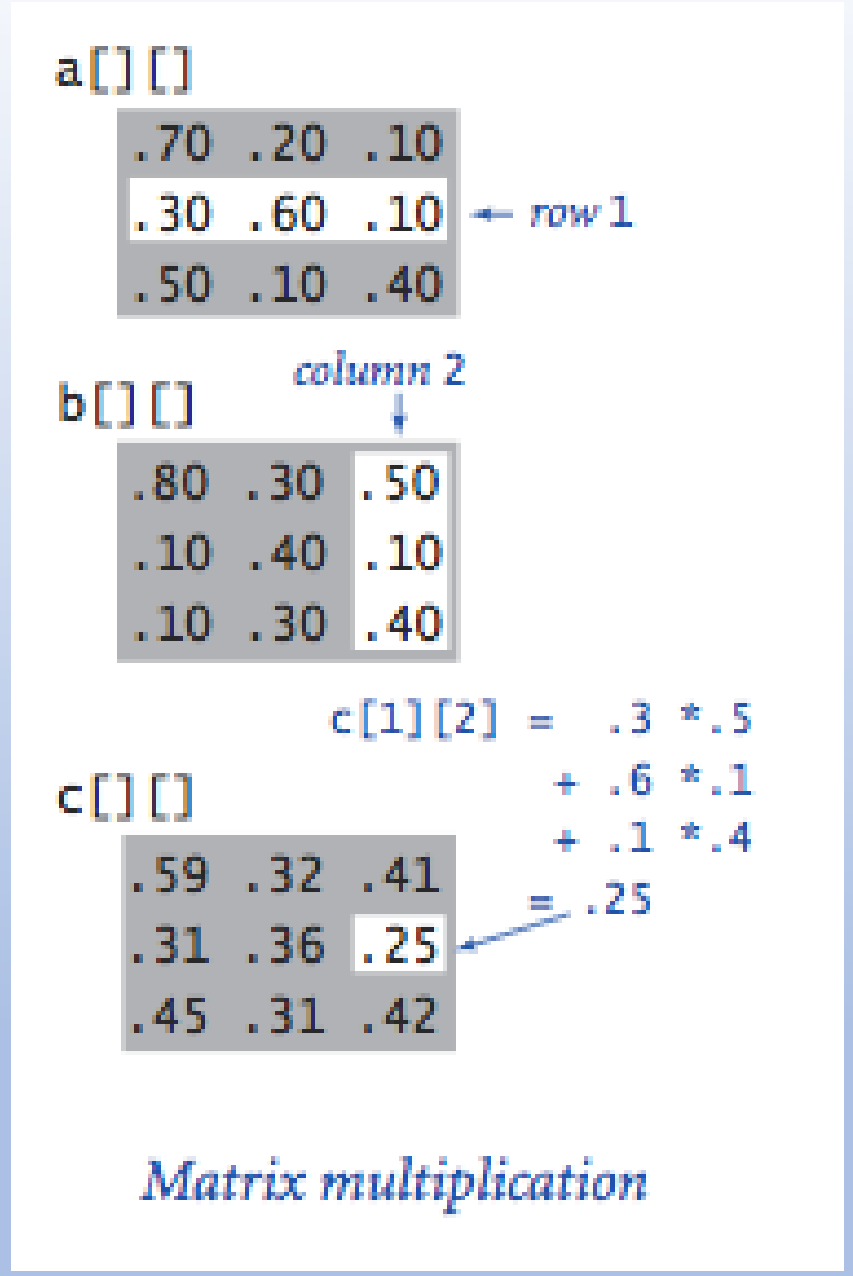
```
a[i][j][k]
```

Matrix operations

- Exercise
- Write code to add 2 n x n matrices.
- Write code to multiply 2 n x n matrices.

Add

```
double[][] c = new double[n][n];
for (int i = 0; i < n; i++) {
    for (int j = 0; j < n; j++) {
        c[i][j] = a[i][j] + b[i][j];
    }
}
```



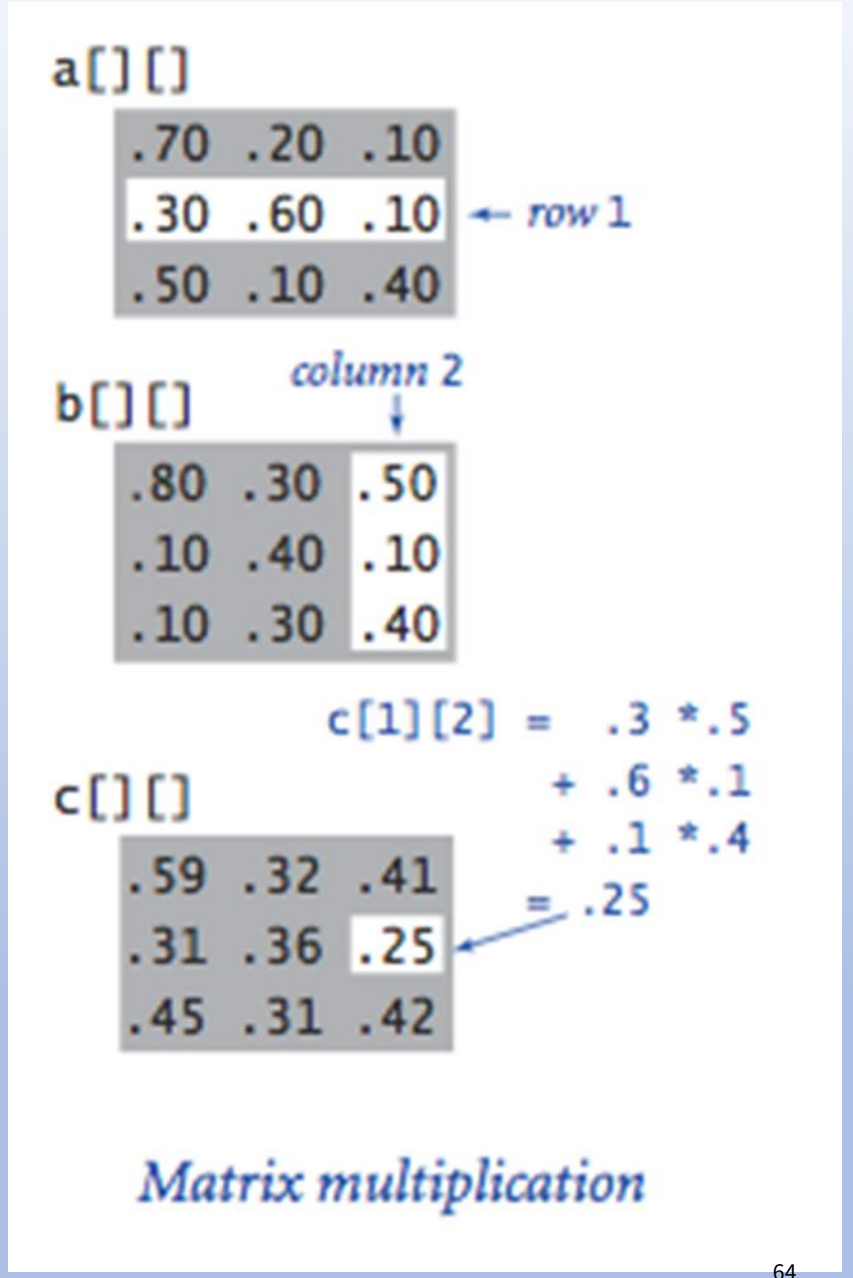
Matrix multiplication

Multiply

Each entry `c[i][j]` in the product of `a[]` and `b[]` is computed by taking the dot product of row `i` of `a[]` with column `j` of `b[]`

Multiply

```
double[][] c = new double[n][n];
for (int i = 0; i < n; i++) {
   for (int j = 0; j < n; j++)  {
      for (int k = 0; k < n; k++)  {
         c[i][j] += a[i][k]*b[k][j];
      }
   }
}
```

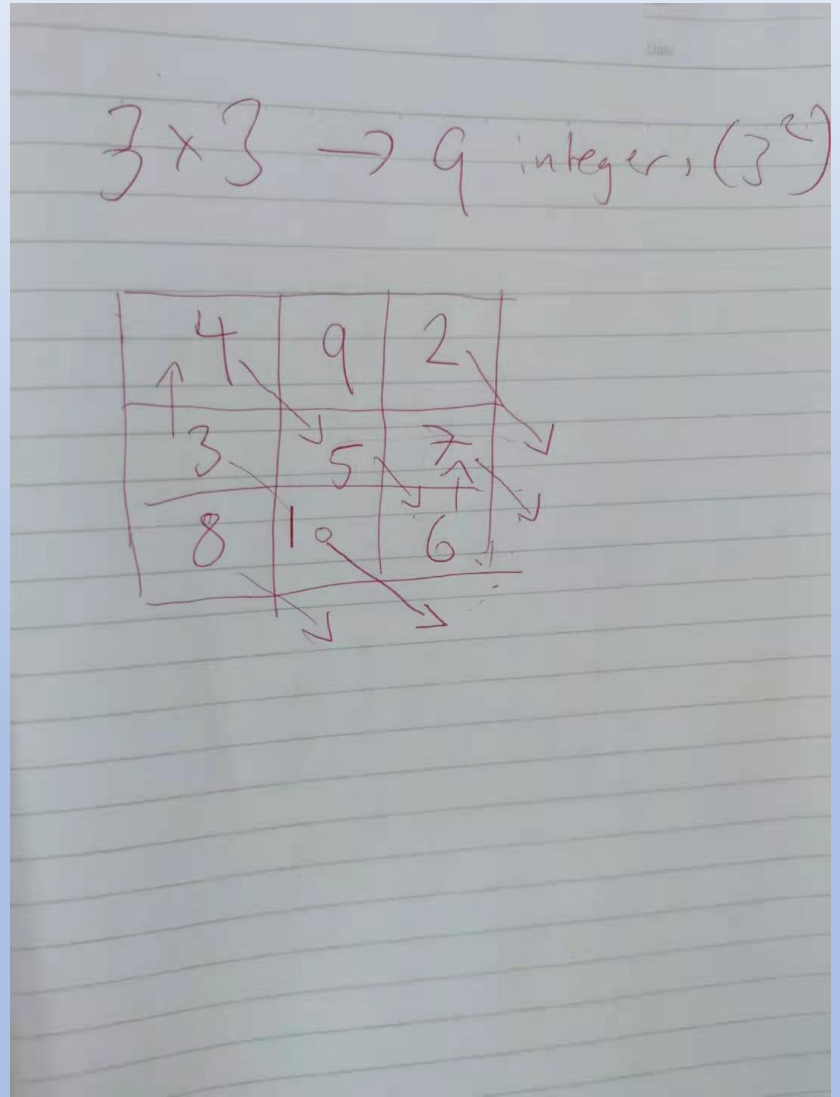


Magic Square

A n by n magic square is an n x n matrix with the integer 1 .. n^2 exactly once, with the constraints that the rows, columns and diagonals all are equal.

One simple algorithm is to assign the integers 1 to n^2 in ascending order, starting at the bottom, middle cell. Repeatedly assign the next integer to the cell adjacent diagonally to the right and down. If this cell has already been assigned another integer, instead use the cell adjacently above. Use wrap-around to handle border cases.

One simple algorithm is to assign the integers 1 to n^2 in ascending order, starting at the bottom, middle cell. Repeatedly assign the next integer to the cell adjacent diagonally to the right and down. If this cell has already been assigned another integer, instead use the cell adjacently above. Use wrap-around to handle border cases.



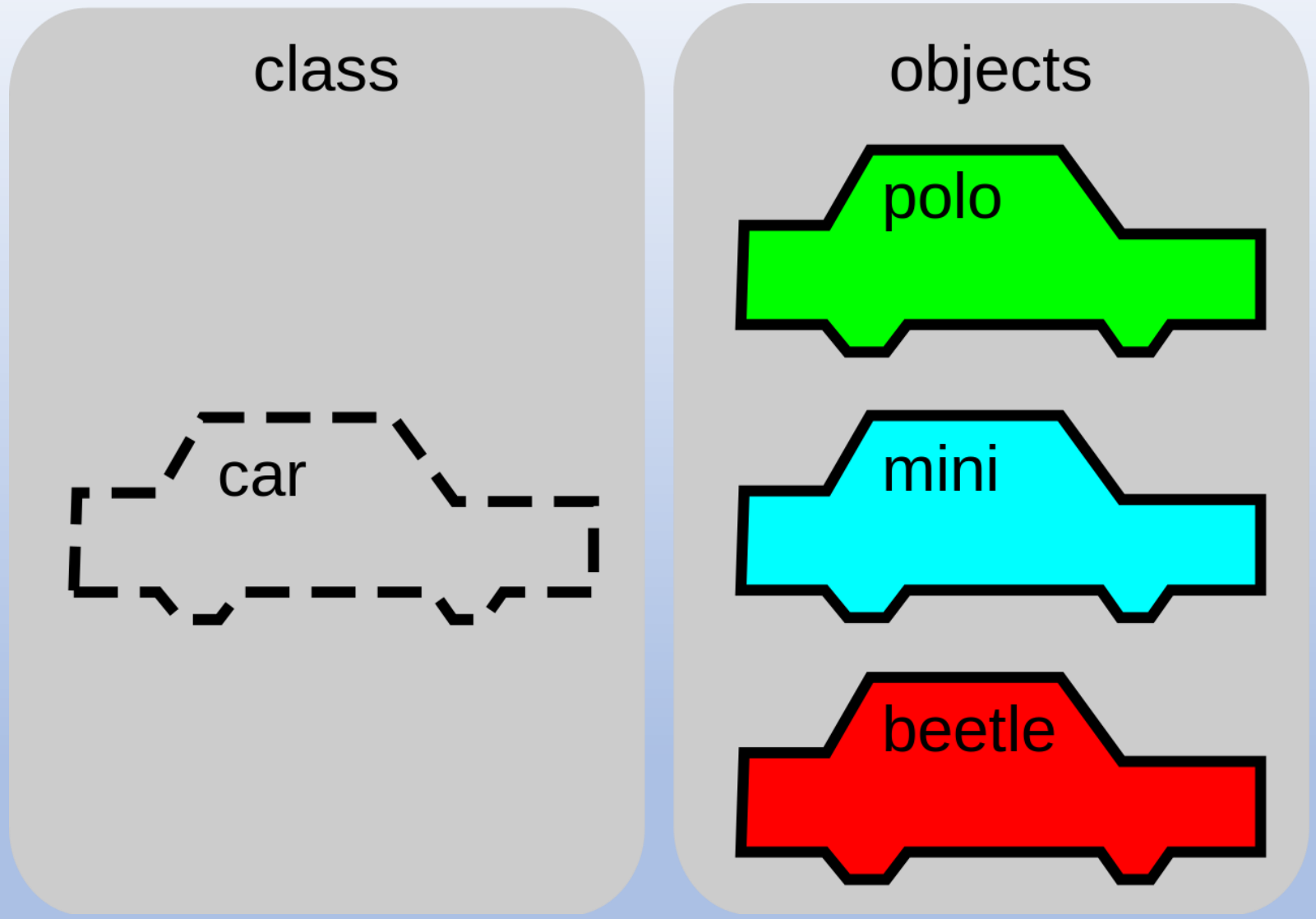
```
int[][] magic = new int[n][n];

int row = n-1;
int col = n/2;
magic[row][col] = 1;

for (int i = 2; i <= n*n; i++) {
    if (magic[(row + 1) % n][(col + 1) % n] == 0) {
        row = (row + 1) % n;
        col = (col + 1) % n;
    }
    else {
        row = (row - 1 + n) % n;
        // don't change col
    }
    magic[row][col] = i;
}
```

```java
public class MagicSquare {

    public static void main(String[] args) {
        int n = Integer.parseInt(args[0]);
        if (n % 2 == 0) throw new RuntimeException("n must be odd");

        int[][] magic = new int[n][n];

        int row = n-1;
        int col = n/2;
        magic[row][col] = 1;

        for (int i = 2; i <= n*n; i++) {
            if (magic[(row + 1) % n][(col + 1) % n] == 0) {
                row = (row + 1) % n;
                col = (col + 1) % n;
            }
            else {
                row = (row - 1 + n) % n;
                // don't change col
            }
            magic[row][col] = i;
        }

        // print results
        for (int i = 0; i < n; i++) {
            for (int j = 0; j < n; j++) {
                if (magic[i][j] < 10)  System.out.print(" ");  // for alignment
                if (magic[i][j] < 100) System.out.print(" ");  // for alignment
                System.out.print(magic[i][j] + " ");
            }
            System.out.println();
        }

    }
}
```

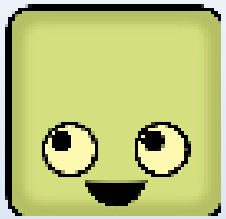
OOP

Object Oriented Programming

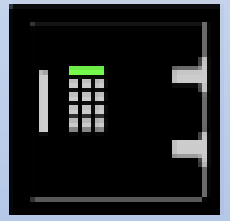
- Java is especially suited to object oriented programming paradigms
- "Object Orientation" can be contrasted with procedural (or functional) orientation
- When you start with programming, you are really concerned about "if" statements and loops, you are happy with having a program that compiles and gives the expected result, but usually you don't go much further. With experience, you realize that you may have many ways to organize a program. The first programming languages were purely "procedural" languages, describing steps to perform to obtain the result. Object orientation brought a new way of looking at things. There are still other ways and other kind of languages, such as declarative languages where you simply state what you want (database queries) and functional languages. Depending on the problem to solve, one approach may be easier than another one.


class
car
objects
polo
mini
beetle

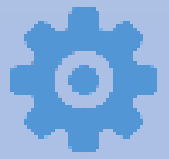
Objects



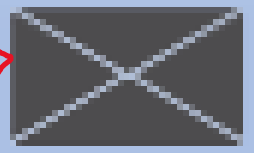
Identity



Object Data



Provides services

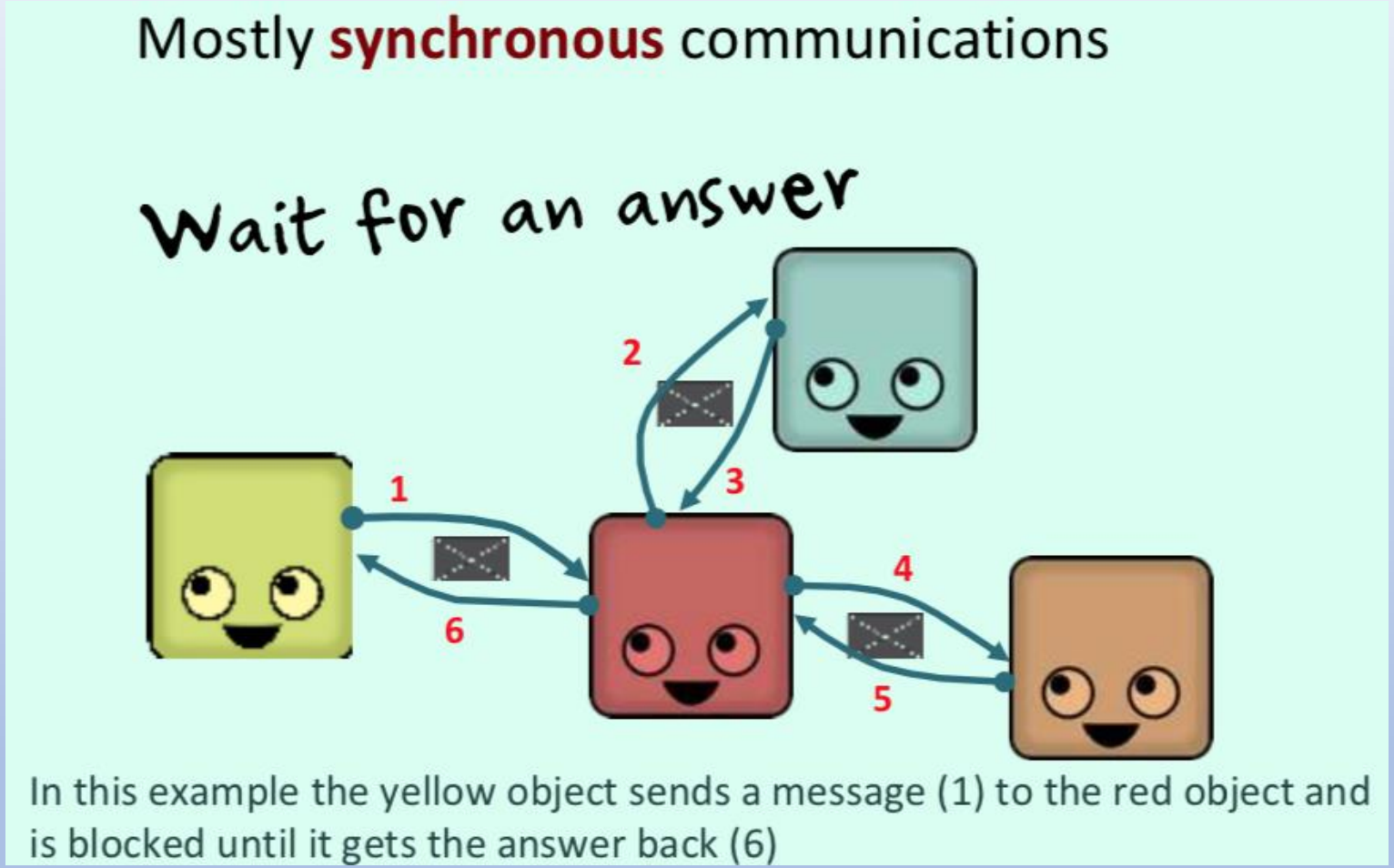


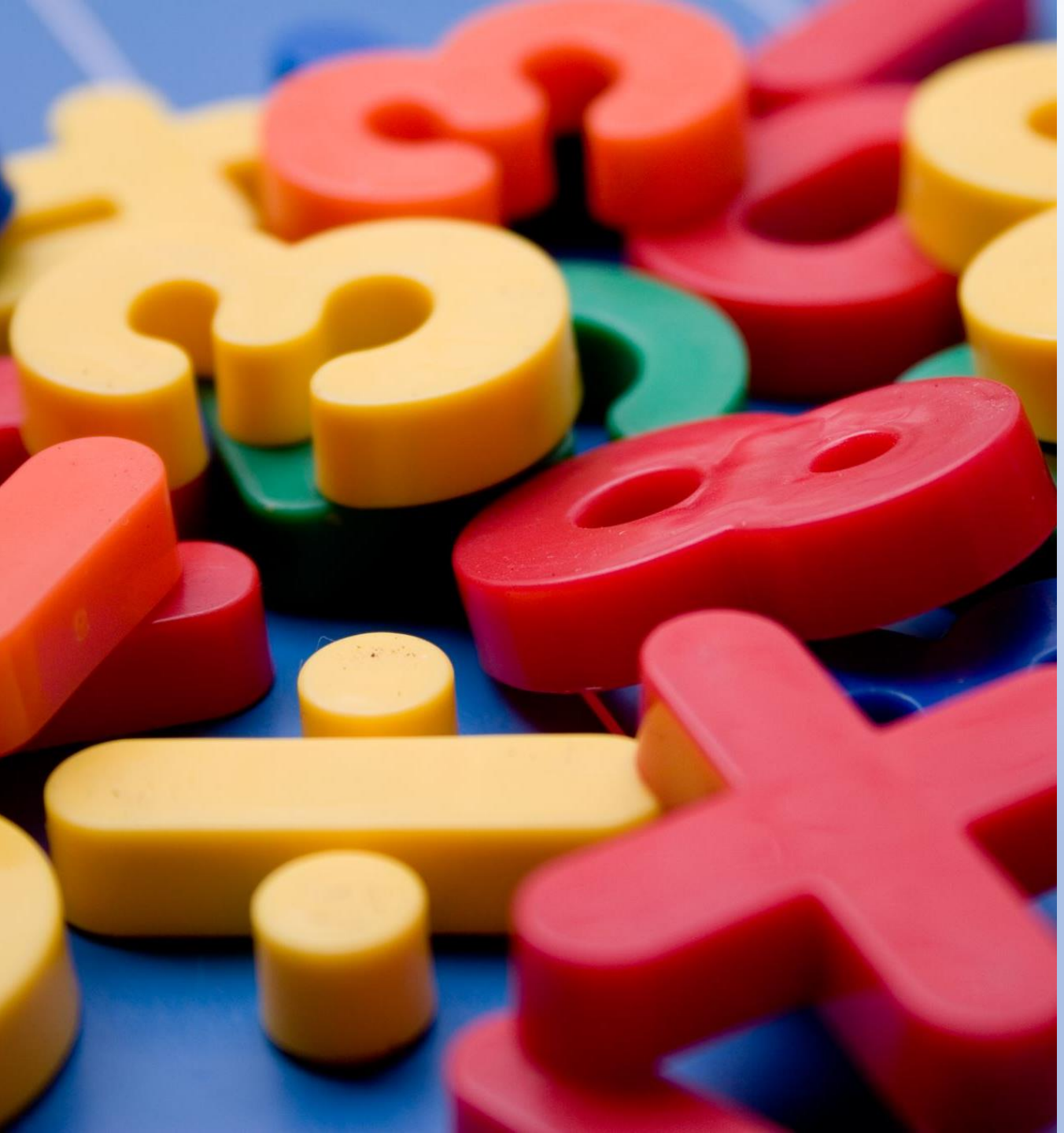
Message

- In Computer Science, an Object is a software component that has an identity, stores data that is usually carefully hidden ("private") and provide services. The object provides services through messages, which are the method calls. Of course, to send a message you must know whom to send it, hence the identity, called "reference" in Java.
- An object can provide services to itself and in this case the identity is “this”, which means “me, this object”

Messaging

When messages are exchanged, they are **synchronous** (a word that comes from Greek and means "same time"), which practically means that the caller waits for the answer and does nothing until it gets it.



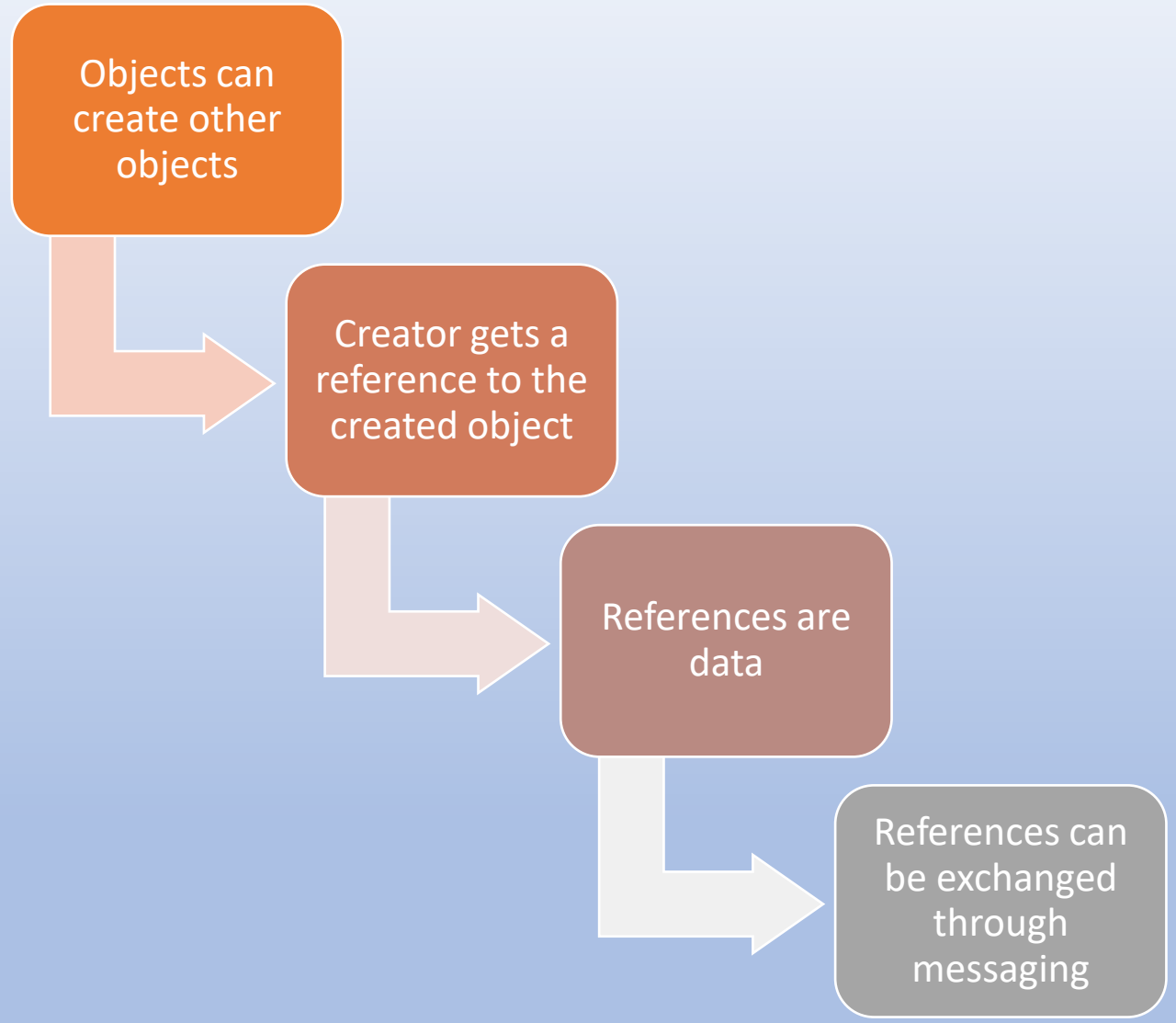


Coupling

- When many objects exchange messages between each other it's said that there is a higher degree "coupling" and it can be very slow (especially when, as it happens sometimes, the objects aren't all running in the same machine).
- On one hand it can be better to divide a big task in many elementary tasks executed by different objects, but on the other it increases the number of messages to exchange.
- Higher coupling = Slower programs

Object References

- You need the identity (reference) of an object to exchange a message (call a method) with it. An object can create another object by invoking its "constructor" and the constructor returns the reference. But the creator may not be the only object that expects services from the newly created object. In many cases, it will pass around the reference (as a method parameter) so that the other objects it requests a service from can use services from this new object.
- It's not recommended to let every object create objects, and it's considered to be a better practice to have objects that act as "object factories" and then pass along references. This is a technique known as "dependency injection".



Public static void main(String args[])

- In Java, there is a very special object with a method called main() that pops out of nowhere to create the first objects in the application and start everything running.
- The special object (*main*) creates the first objects.
- A **bootstrap** is the program that initializes the operating system (OS) during startup.
- The term **bootstrap** or **bootstrapping** originated in the early 1950s. It referred to a **bootstrap** load button that was used to initiate a hardwired **bootstrap** program, or smaller program that executed a larger program such as the OS.

Test.java

```
1  package com.journaldev.java.examples1;
2
3  public class Test {
4
5      public static void main(String args[]) {
6
7          System.out.println("Hello World");
8
9      }
10 }
```

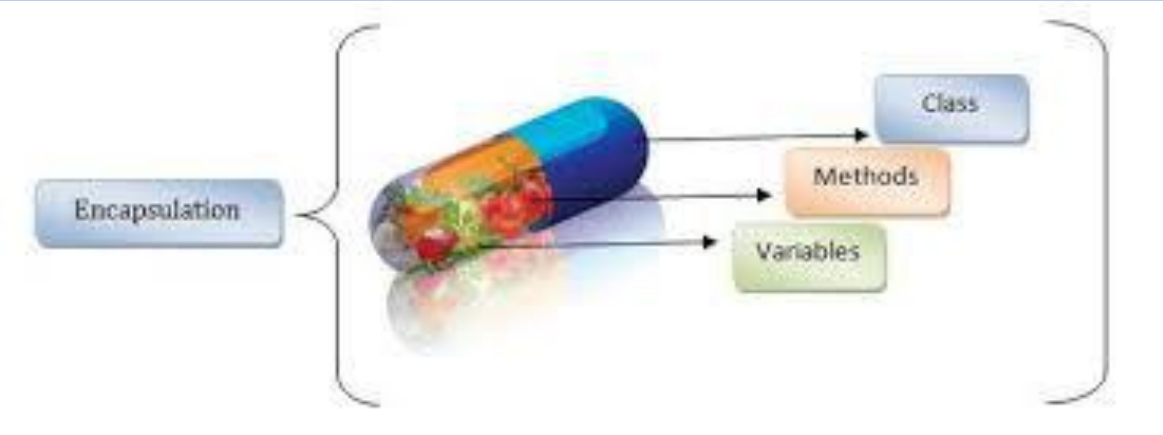
Designing OO Programs

- As I have already hinted, one of the problems in object oriented programming is that some of recommended strategies are at odds with each other. Small objects doing small tasks are better for consistency (which means that objects are focused on one task), reuse, and testing (you can write a mock object that returns hard-coded values while someone is debugging the real thing); but they increase communications and coupling.
- Better to have several small objects than a big one (testing) - consistency
- Better to pass references than leave *any* object create new objects
- Better to minimize communications (weak coupling)

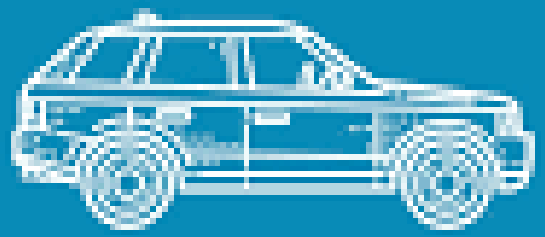
You mustn't forget some of the key principles of object-oriented programming: encapsulation, which means that everything is private except the public methods that define the interface of the object with the outside world, and responsibility.

KEY PRINCIPLE: Encapsulation

Responsibility – the object has all the means to provide the service



Encapsulation



Car

model

speed

engine

speedLimit

drive()

stop()

setSpeed(number)

Object Oriented Programming…

Entities

Having state

Having a behaviour

Exchanging messages

Challenges

- In theory, Object Oriented programming is easy. There is a saying that "the gap between theory and practice is wider in practice than it is in theory". One (but perhaps no the greatest) difficulty is to correctly identify objects. In spite of their name, "objects" don't always correspond to real-life objects; sometimes they implement a function. Let's illustrate it with an example.

Exceptions

Exceptions

```
try {
    line = console.readLine();

    if (line.length() == 0) {
        throw new EmptyLineException("The line read from console was empty!");
    }

    console.printLine("Hello %s!" % line);
    console.printLine("The program ran successfully.");
}
catch (EmptyLineException e) {
    console.printLine("Hello!");
}
catch (Exception e) {
    console.printLine("Error: " + e.message());
}
finally {
    console.printLine("The program is now terminating.");
}
```

lier we looked at how the JVM works. Now we will k at the concept of “Exceptions”.

ava, exceptions are a common way to change the rmal path of execution of a program; they first peared in the 1960s in a language called LISP signed for artificial intelligence applications).

ava, exceptions are often used to control the flow nstruction execution to handle different cases. In + and Lisp, on the other hand, they are used for normal and unpredictable errors.

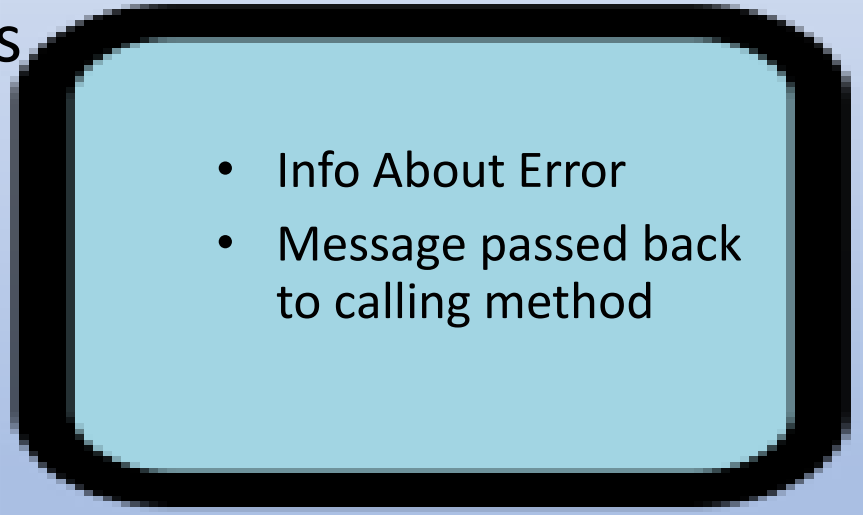
Background Knowledge

- This class assumes some basic familiarity of exceptions from first year CS
- For further details of exceptions and background reading please refer to the section 3.7 in Text B:
  - http://math.hws.edu/eck/cs124/javanotes8/c3/s7.html

Special

Exceptions are Objects

- An exception in Java is a special object, generated by an "exceptional event" (understand, most often, an error) that is passed back to the calling method through a mechanism different from the usual messaging between objects.



Throw: Trigger an Exception

- You throw an object, not a type
- throw new MyException();
- As an exception is an object, is must be instantiated (created from the template that a class defines as a type) using **new** when you need it and **throw** it.

Types of Errors

- Compile Time
  - Syntax Errors
  - Wrong Type

- Many things can go wrong, so we have different types of errors. Javac, as it compiles .java files to .class java bytecode files, will tell you about wrong syntax, or incompatible types when assigning data or calling methods. You have to solve all these issues before you can think about running your program.

Types of Errors

- Compile Time
- Link Time

- Before you can run your program, the Class Loader must load it and you may have passed compilation and failed linking, because the Class Loader fails to find a .class corresponding to objects you want to use.

Types of Errors

- Compile Time
- Link Time
- Run Time
  - Detected by the application
  - Detected by the library
  - Detected by the Operating System / Hardware

These are exceptions

- When you run your program, you may run into other errors; one of your application methods can detect that it gets parameters that are the right type but the wrong range of values. A built-in Java method may discover the same (and will throw an exception); or things may go really wrong and the operating system may discover it (hardware failure, for instance). Logic can also be wrong.

Types of Errors

- Compile Time
- Link Time
- Run Time
- Logic

- Your programs logic can also be incorrect.

Java Syntax and Runtime

- As some exceptions occur quite often, some methods warn about it by using **throws** followed by the name(s) of the exceptions it can throw. The javac compiler is then made aware of them.
- Methods that throw exceptions say so …
- Class FileReader
- Public FileReader (String filename) throws FileNotFoundException
- …
- … So javac knows about it.

Java Syntax and Runtime

```
import java.io.File;
import java.io.FileReader;
public class IgnoredException {
    public static void main(String args[]) {
        FileReader fr = new FileReader("sample.txt");
    }
}
```

If you happily ignore the exceptions advertised by a method such as the previous constructor ...

Java Syntax and Runtime

- $ javac IgnoredException.java
  IgnoredException.java:7: error: unreported exception
  FileNotFoundException; must be caught or declared to be thrown
- FileReader fr = new FileReader("sample.txt");
- ^
- 1 error
- $

... javac will simply not let you do it. The message is pretty explicit: it wants you to either deal with the problem, or let the world know that something may fail and that another object has to deal with it.

So there are 2 Options

- Deal with it (try … catch …)

You must either include the method call that can fail into a try ... catch block and deal with the exception in the catch block.

So there are 2 Options

- Warn callers (throws …)

Or you may get away with handling the exception if you warn callers that what you are doing for them may fail, and that in that case *they* will have to deal with the problem.

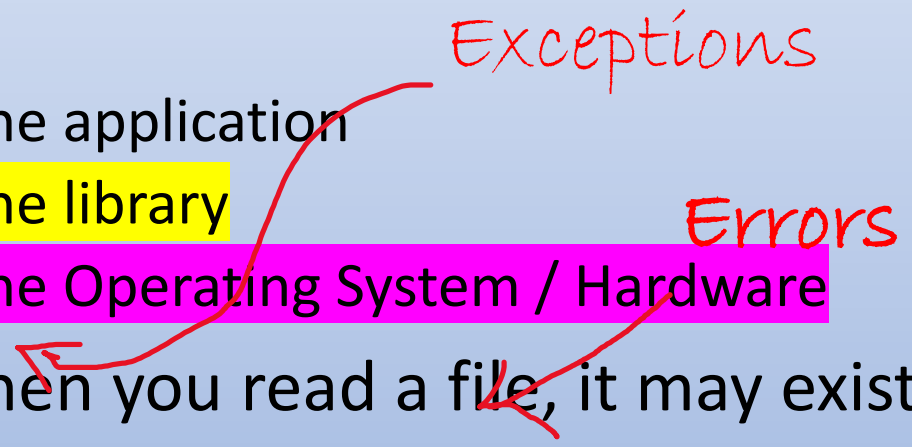
Checked Exception

- Handled
- OR
- Declared as thrown

- When you are faced with this choice you are dealing with what is known as a checked exception, and javac will make sure that you follow the rules.

But something else can happen:

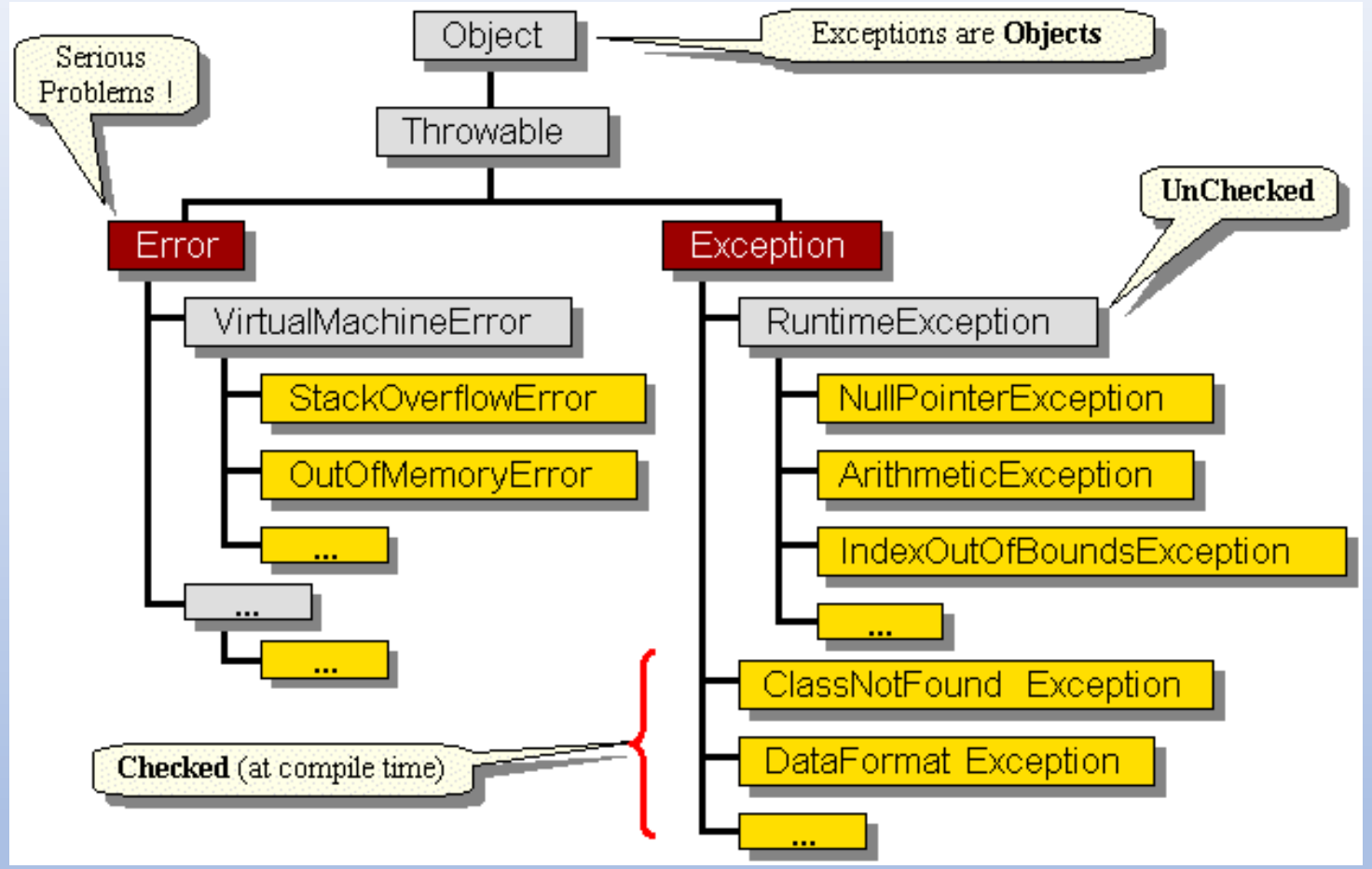
- Compile Time
- Link Time
- Run Time
  - Detected by the application
  - Detected by the library
  - Detected by the Operating System / Hardware
- For example when you read a file, it may exist but be corrupted.



java.io.IOError

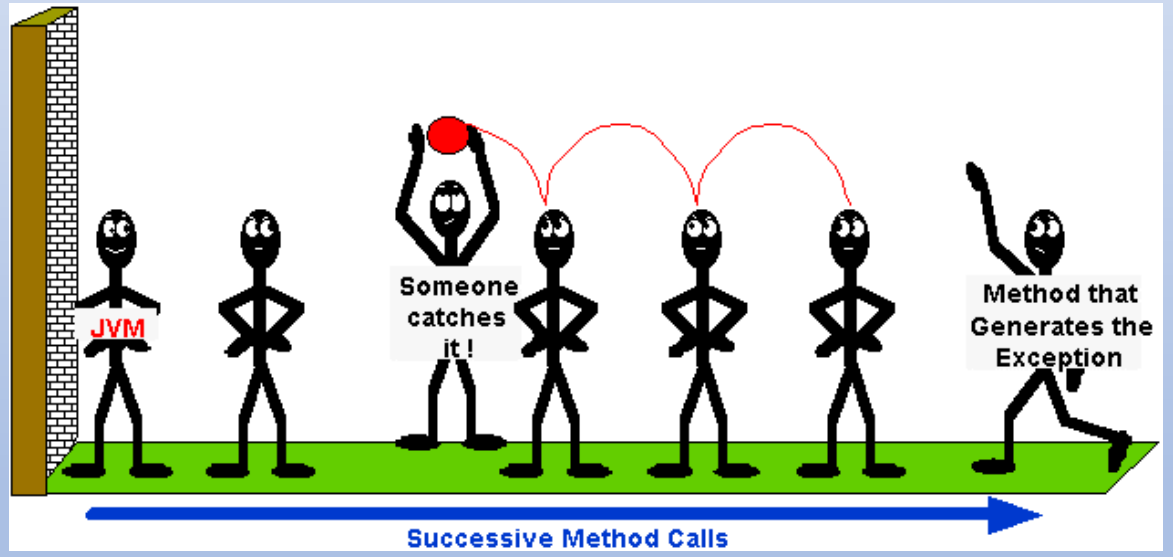
Exceptions VS Errors

- Exceptions are abnormal conditions that occur in your program or java telling you something has gone wrong. When an exception occurs java forces us to handle it (need to know when and what to do) or declare we want someone else to handle it.
- Exception handling provides a way to handle and recover from errors or a way to quit the program in a graceful way.
- Errors represent serious represent serious unrecoverable problems for example VirtualMachineError.OutOfMemoryError



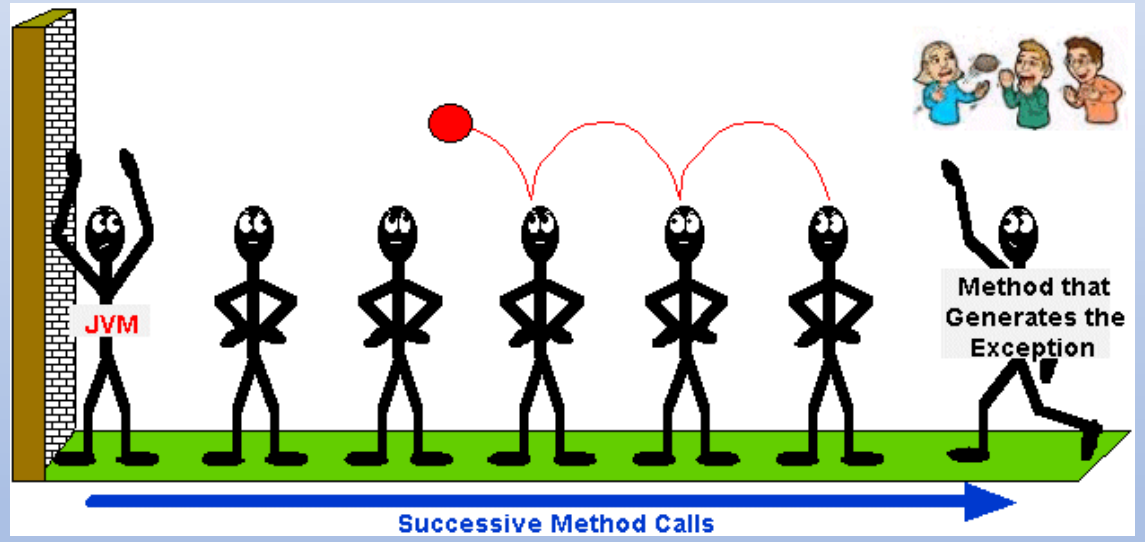
Exceptions VS Errors

Throwing Exceptions



- …
- public void getInformation() throws MyException {
- …
- }
- private void doStuff(){
- try {
- getInformation();
- catch (MyException ex){
- //handle here
- }

Throwing Exceptions



- …
- public void getInformation() throws MyException {
- …

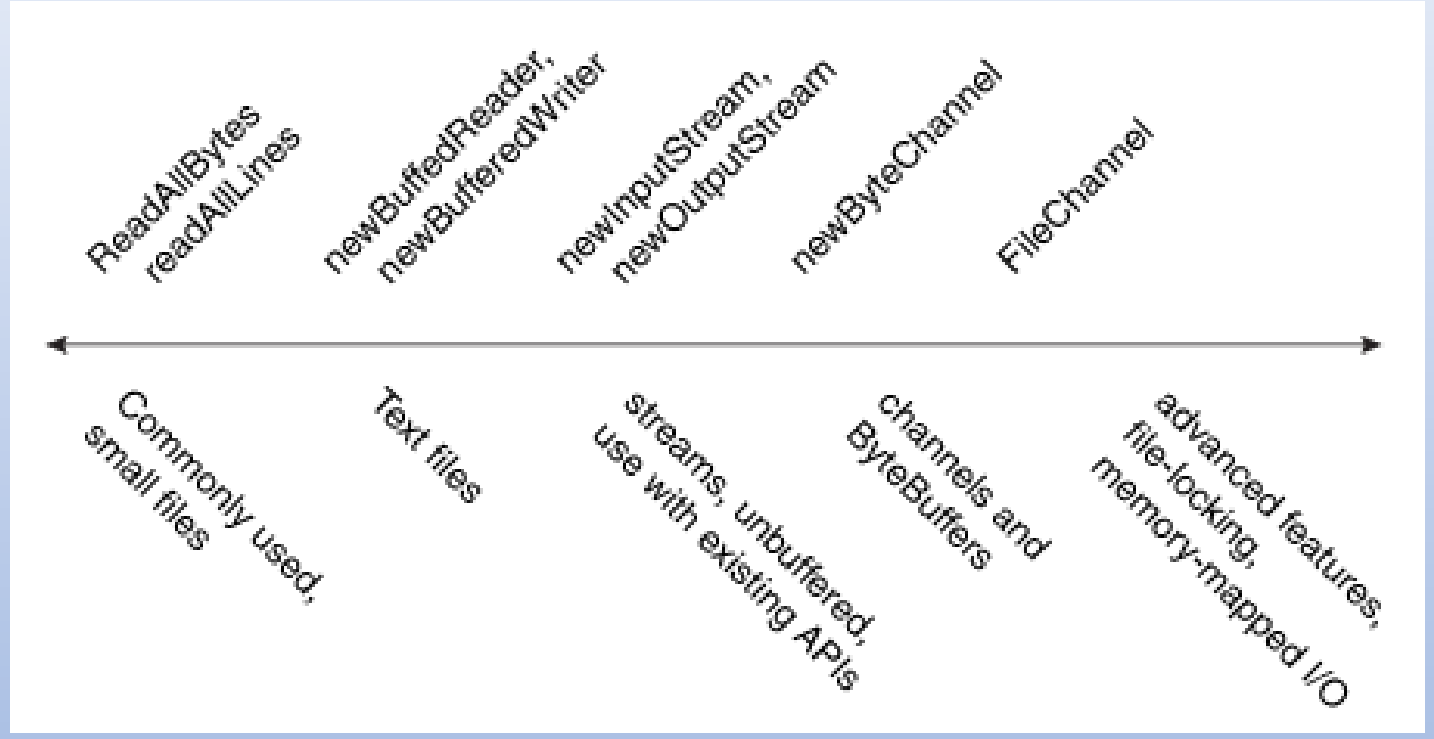
Lab

File IO

- In first year you used files to read and save data on a disk
- You can review the basics of file IO here: https://docs.oracle.com/javase/tutorial/essential/io/file.html

File IO

- There are many file I/O methods to choose from. Here are the file I/O methods arranged by complexity (left is simpler)
- In the lab we emphasize the use of exception handling to facilitate reading and writing from files in Java



Internationalization

- Internationalization refers to making programs that can take input and output that is tailored to different locations and languages.
- A character encoding can take various forms depending upon the number of characters it encodes. The number of characters encoded has a direct relationship to the length of each representation which typically is measured as the number of bytes. **Having more characters to encode essentially means needing lengthier binary representations.**
- Historically the first character encodings were used to map between telegraph signals and letters in a minimal way (often not considering case), here encoding refers to a mapping from characters used in language to 0s and 1s used in binary representation. As different languages contain a wide variety of letters and characters, for example Chinese, Arabic, English, etc, character sets are an important element of in the process of making software systems international or language/location independent.

- For further examples see https://docs.oracle.com/javase/tutorial/i18n/intro/quick.html

Character Encoding

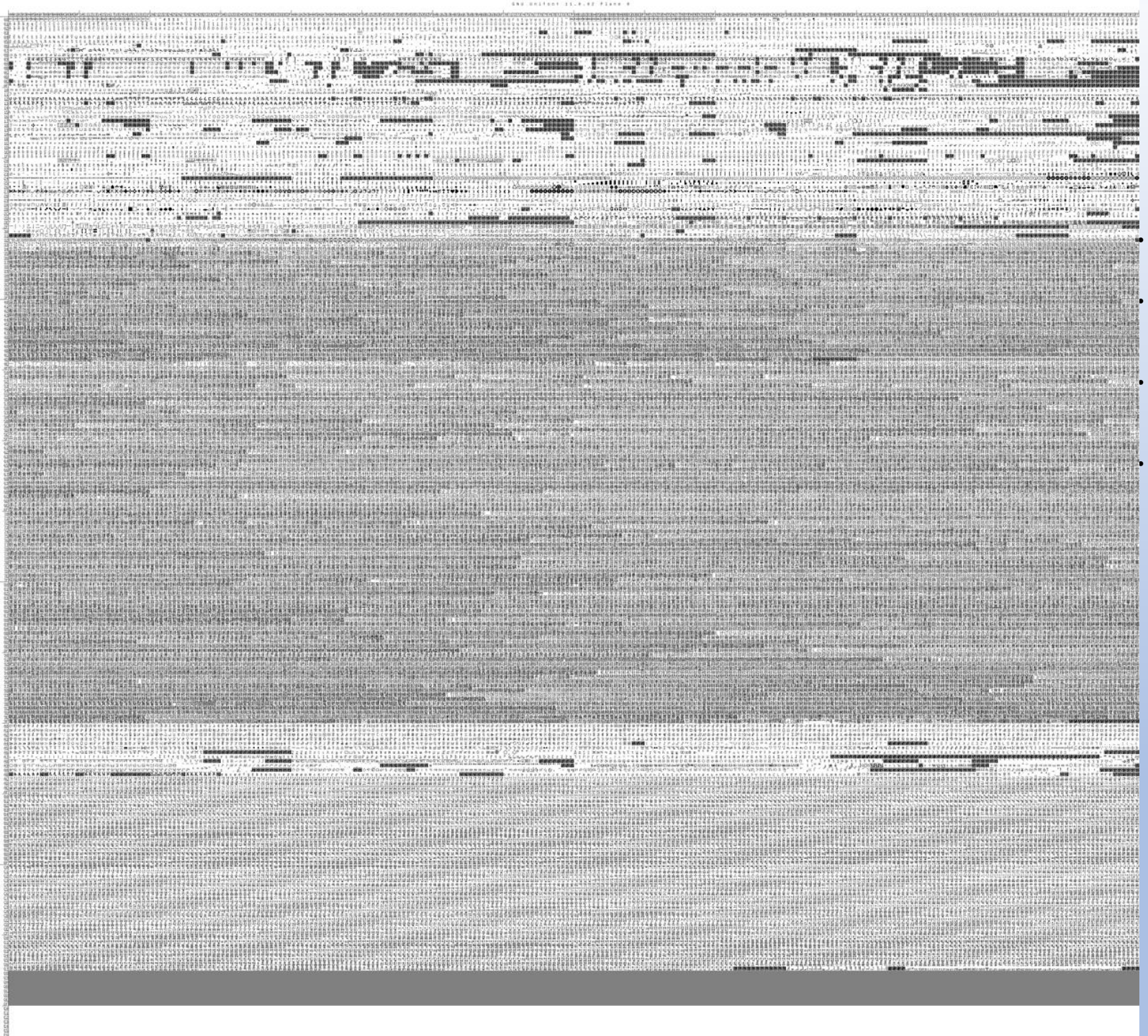
- The International Morse Code encodes the 26 English letters A through Z, some non-English letters, the Arabic numerals and a small set of punctuation and procedural signals (prosigns). There is no distinction between upper and lower case letters.

International Morse Code

- 1 dash = 3 dots.
- The space between parts of the same letter = 1 dot.
- The space between letters = 3 dots.
- The space between words = 7 dots.

A	·−	V	···−
B	−···	W	·−−
C	−·−·	X	−··−
D	−··	Y	−·−−
E	·	Z	−−··
F	··−·	.	·−·−·−
G	−−·	,	−−··−−
H	····	?	··−−··
I	··	/	−··−·
J	·−−−	@	·−−·−·
K	−·−	1	·−−−−
L	·−··	2	··−−−
M	−−	3	···−−
N	−·	4	····−
O	−−−	5	·····
P	·−−·	6	−····
Q	−−·−	7	−−···
R	·−·	8	−−−··
S	···	9	−−−−·
T	−	0	−−−−−
U	··−		

Chinese telegraph code



Unicode

- Different encoding schemes developed in isolation and practiced in local geographies started to become challenging.
- It is not difficult to understand that while encoding is important, decoding is equally vital to make sense of the representations. **This is only possible in practice if a consistent or compatible encoding scheme is used widely.**
- This challenge gave rise to **a singular encoding standard called Unicode which has the capacity for every possible character in the world**. This includes the characters which are in use and even those which are defunct!
- **Therefore, how these code points are encoded into bits is left to specific encoding schemes within Unicode.**

Unicode encodings

- UTF-32 is **an encoding scheme for Unicode that employs four bytes to represent every code point** defined by Unicode. Obviously, it is space inefficient to use four bytes for every character.
  - For example: "T” in "UTF-32” is "00000000 00000000 00000000 01010100” (the first 3 bytes are unnecessary)
- Variable length encoding: UTF-8 is **another encoding scheme for Unicode which employs a variable length of bytes to encode**. While it uses a single byte to encode characters generally, it can use a higher number of bytes if needed, thus saving space.
  - For example: "T” in "UTF-8” is "01010100”
  - But "語” in "UTF-8” is "11101000 10101010 10011110"

Summary

- FileIO, Internationalization, and Charset encoding are important practical concepts

- In this weeks practical we look at some examples and demonstrate how topics from the lectures including Object Orientation and Exception Handling are able to be used in these problems