بنام خداوند جان و خرد

Data Structures and Algorithms

Soheila Ashkezari-T.

@SAshkezari

Lecture 1: Intro to ADTs

Introduction

- ساختمان داده‌ها و الگوریتم‌ها درسی ۴ واحدی و جزء دروس اصلی در رشته علوم کامپیوتر می‌باشد.
- با توجه به اهمیت درس و ارائه مطالب متنوع، اکیدا توصیه می‌گردد در همه جلسات درس شرکت داشته باشید.

- مراجع:

- Data structures and algorithms in java™, sixth edition, michael t. Goodrich, roberto tamassia, michael h. Goldwasser, wiely, 2014
- Data Structures and Algorithms in C++ 2nd Edition, Michael T. Goodrich, Roberto Tamassia, David M. Mount, Goodrich, Michael T., Roberto Tamassia, John Wiley & Sons, 2011.
- Data structures abstraction and design using java, elliot b. Koffman, paul a. T. Wolfgang, wiley, 2016
- Algorithms, Robert Sedgewick, kevin Wayne, Princeton University, 2011

Introduction

- مرجع اصلی اسلایدها:
  - درس ساختمان داده‌ها و الگوریتم‌ها، دانشگاه واشنگتن، زمستان ۲۰۲۴
  - https://courses.cs.washington.edu/courses/cse373/24wi/

  - در برخی موارد تغییراتی اعمال شده است.
  - نکاتی از کتاب به اسلایدها اضافه شده است.

Introductions

Syllabus

Basic Concept

Questions

Course Overview

Course Goals

- Design data structures and algorithms by implementing and maintaining invariants.
- Analyze the runtime and design data structures and algorithms
- Critique the application of data structures and algorithms towards complex problems

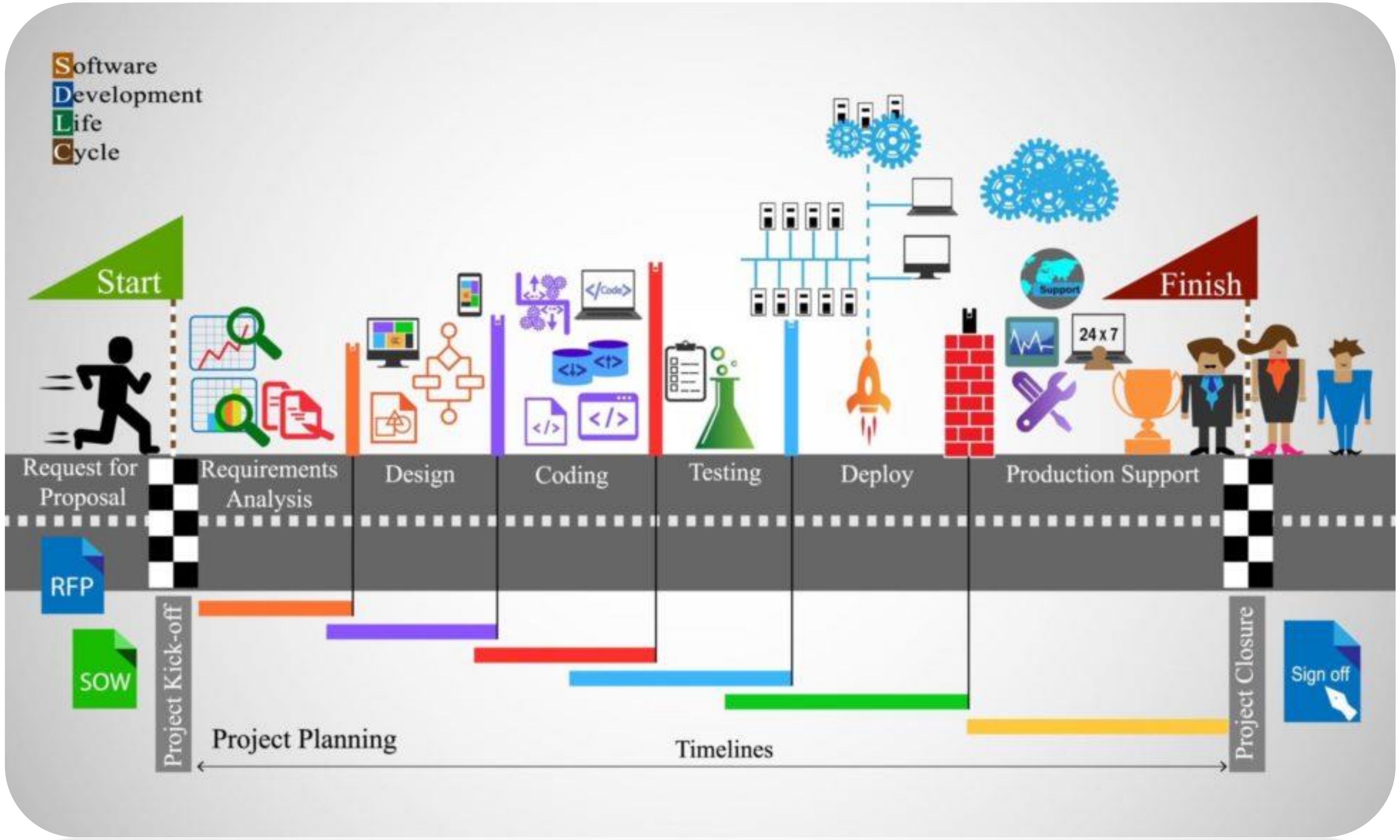
Course Topics

- Data Structures and ADTs: lists, stacks, queues, sets, dictionaries, arrays, linked lists, trees, hash tables, priority queues, binary heaps and disjoint sets
- Algorithm analysis: Big O Notation, asymptotic analysis, P and NP complexity
- Sorting algorithms: selection, insertion, merge, quick, more…
- Graphs and graph algorithms: graph search, shortest path, minimum spanning trees

Grading Breakdown

- **آزمون میان ترم اول**
  - ۴ نمره
  - ۲۸ آبان ۱۴۰۳
- **آزمون میان ترم دوم**
  - ۴ نمره
  - ۳ دی ۱۴۰۳
- **آزمون پایان ترم**
  - ۴ نمره
  - روز دهم امتحانات
- **تمرین و پروژه (۱۰ نمره):**
  - پروژه اول: ۳ نمره
  - پروژه دوم: ۳ نمره
  - تمرین ها: ۳ نمره
  - ارائه انفرادی (اختیاری): ۱ نمره
  - پروژه ها به صورت گروه دو نفری و تمرین ها انفرادی است.
  - کشف تقلب یا عدم تسلط در هر کدام از بخش‌های فوق به منزله عدم کسب نمره آن بخش می باشد.
- کسب **حداقل ۵ نمره از مجموع آزمون های کتبی** برای گذراندن درس الزامی می باشد.

The Software Development Life Cycle (SDLC)

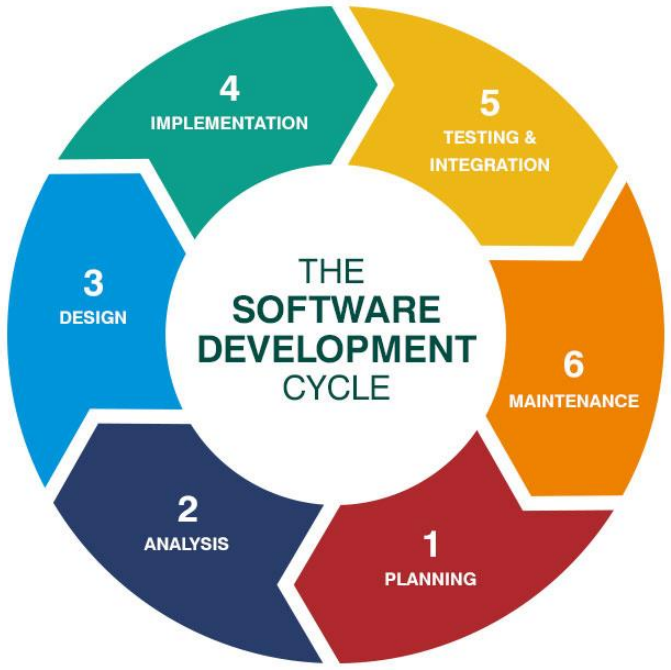


The Software Development Life Cycle (SDLC)

Three major steps which are important in this lecture:

3. Design

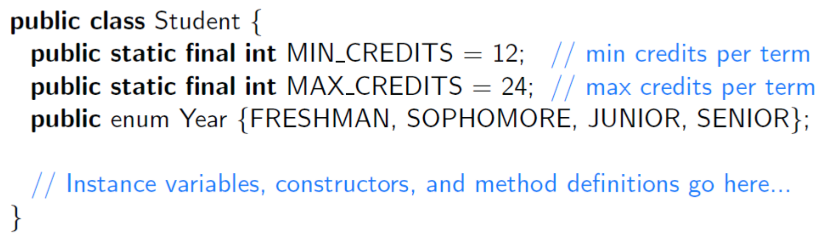
- the most important phase
- Decide how to divide the workings of our program into classes, when we decide how these classes will interact, what data each will store, and what actions each will perform.
- ***Responsibilities***: Divide the work into different ***actors***, each with a different responsibility. Try to describe responsibilities using action verbs. These actors will form the classes for the program.
- ***Independence***: Define the work for each class to be as independent from other classes as possible. Subdivide responsibilities between classes so that each class has autonomy over some aspect of the program. Give data (as instance variables) to the class that has jurisdiction over the actions that require access to this data.
- ***Behaviors***: Define the behaviors for each class carefully and precisely, so that the consequences of each action performed by a class will be well understood by other classes that interact with it. These behaviors will define the methods that this class performs, and the set of behaviors for a class form the ***protocol*** by which other pieces of code will interact with objects from the class.



The Software Development Life Cycle (SDLC)

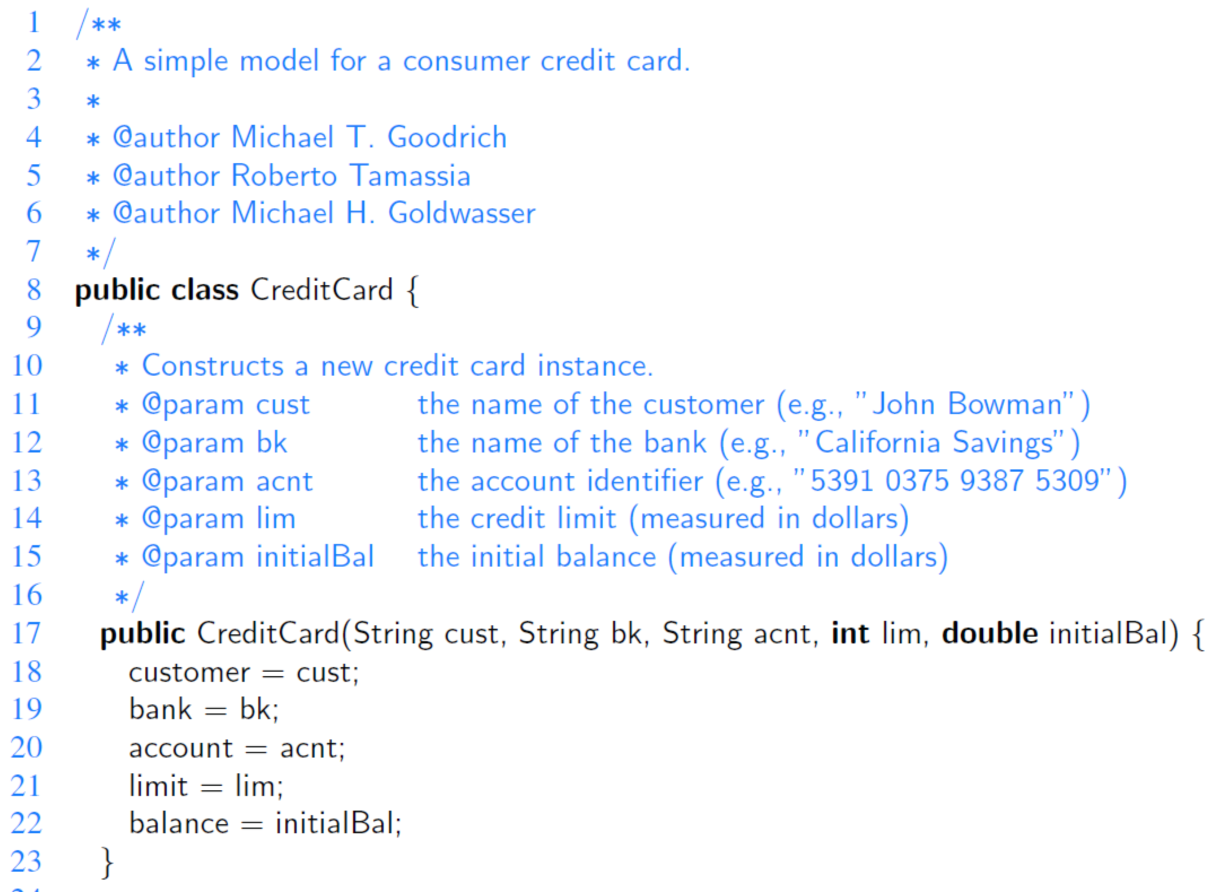
4. Coding/Implementation

- Using Pseudocode
  - It is a mixture of natural language and high-level programming constructs that describe the main ideas behind a generic implementation of a data structure or algorithm.
- Javadoc
  - In order to encourage good use of block comments and the automatic production of documentation, the Java programming environment comes with a documentation production program called javadoc. Each javadoc comment is a block comment that starts with “/**” and ends with “*/” and begin with javadoc tags like @author, @throws, @param, @return
- Readability and Programming Conventions
  - Use meaningful names for identifiers
  - Use named constants or enum types instead of literals.



```
public class Student {
  public static final int MIN_CREDITS = 12;   // min credits per term
  public static final int MAX_CREDITS = 24;   // max credits per term
  public enum Year {FRESHMAN, SOPHOMORE, JUNIOR, SENIOR};

  // Instance variables, constructors, and method definitions go here...
}
```



```
/**
 * A simple model for a consumer credit card.
 *
 * @author Michael T. Goodrich
 * @author Roberto Tamassia
 * @author Michael H. Goldwasser
 */
public class CreditCard {
  /**
   * Constructs a new credit card instance.
   * @param cust         the name of the customer (e.g., "John Bowman")
   * @param bk           the name of the bank (e.g., "California Savings")
   * @param acnt         the account identifier (e.g., "5391 0375 9387 5309")
   * @param lim          the credit limit (measured in dollars)
   * @param initialBal   the initial balance (measured in dollars)
   */
  public CreditCard(String cust, String bk, String acnt, int lim, double initialBal) {
    customer = cust;
    bank = bk;
    account = acnt;
    limit = lim;
    balance = initialBal;
  }
```

The Software Development Life Cycle (SDLC)

4. Coding/Implementation (Continue)

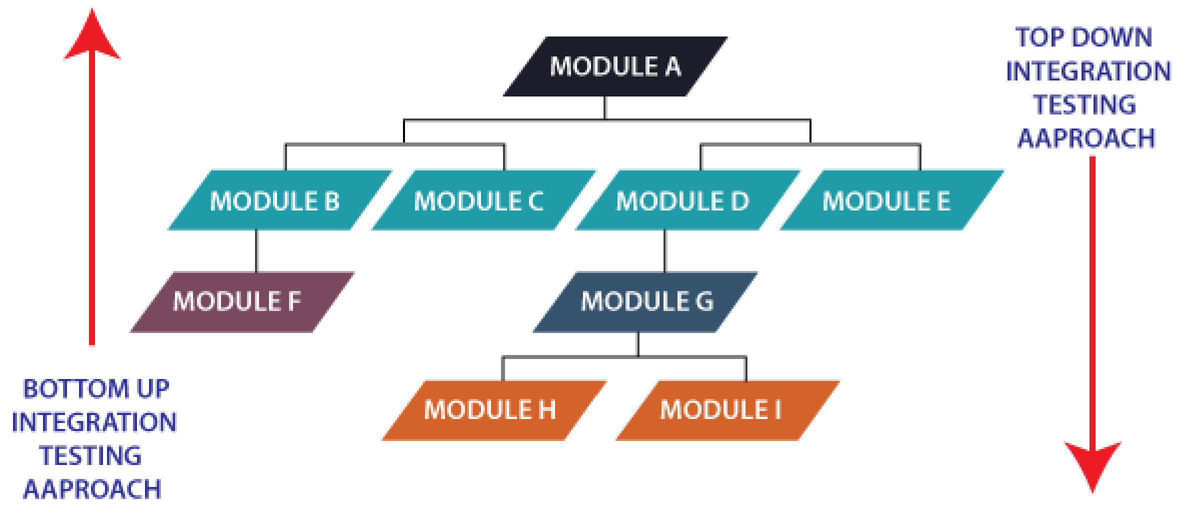
- Indent statement blocks
- Organize each class in the following order:
  - 1. Constants
  - 2. Instance variables
  - 3. Constructors
  - 4. Methods
- Use comments that add meaning to a program and explain ambiguous or confusing constructs.

The Software Development Life Cycle (SDLC)

5. Testing and Debugging

- Testing is the process of experimentally checking the correctness of a program, while debugging is the process of tracking the execution of a program and discovering the errors in it. Testing and debugging are often the most time-consuming activity in the development of a program.
  - ***top-down*** testing (***stubbing)***
  - ***bottom-up*** testing (***unit testing)***

- Debugging. The simplest debugging technique consists of using ***print statements*** to track the values of variables during the execution of the program.
- A better approach is to run the program within a ***debugger.*** The basic functionality provided by a debugger is the insertion of breakpoints within the code.

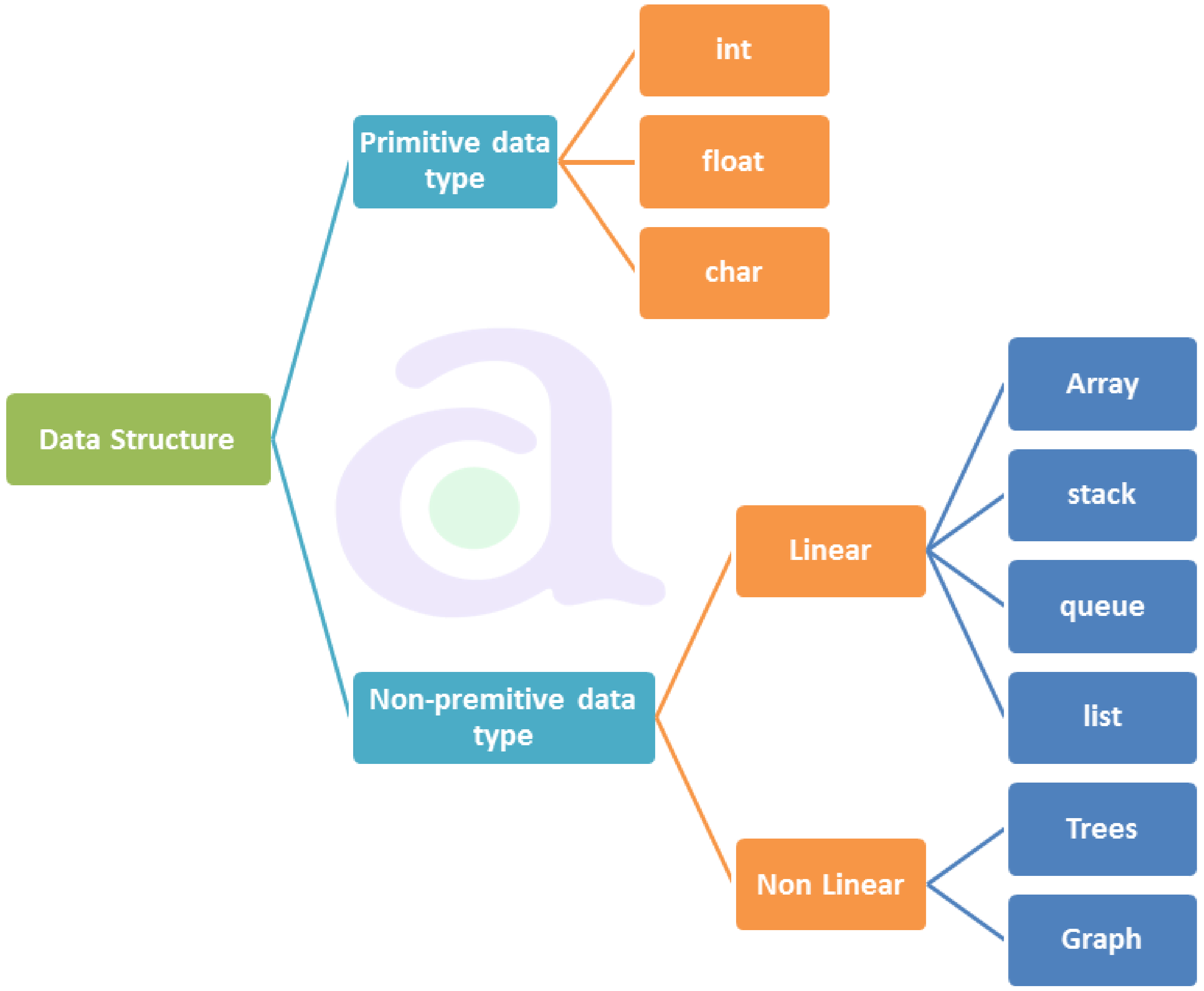

TOP DOWN INTEGRATION TESTING AAPROACH
MODULE A
MODULE B
MODULE C
MODULE D
MODULE E
MODULE F
MODULE G
MODULE H
MODULE I
BOTTOM UP INTEGRATION TESTING AAPROACH

Basic Definitions

Data Structure

- A way of organizing and storing data
- An object that implements the functionality of a specified ADT
- Describes exactly how the collection will perform the required operations
- Even an integer or floating point number stored on the computer can be viewed as a simple data structure.
- Examples: arrays, linked lists, stacks, queues, trees

Data Structure = Data Representation + Legal Operations


Data Structure
Primitive data type
int
float
char
Non-premitive data type
Linear
Array
stack
queue
list
Non Linear
Trees
Graph

Basic Definitions

Type

- **A type is a collection of values:**
- For example, the Boolean type consists of the values true and false.
- An integer is a **simple type** because its values contain no subparts.
- A bank account record will typically contain several pieces of information such as name, address, account number, and account balance. Such a record is an example of an **aggregate type** or **composite type**.
- A **data item** is a piece of information or a record whose value is drawn from a type. A data item is said to be a **member** of a type.

Type: Simple or Composite
= Collections of Values

Basic Definitions

Data Type

- A **data type** is a type together with a collection of operations to manipulate the type.
- For example, an integer variable is a member of the integer data type.
- Addition is an example of an operation on the integer data type.

Data Type = Type + Operations

Basic Definitions

Abstract Data Type (ADT) نوع داده انتزاعی

- A type of organization of data that we define through its behavior and operations
  - Defines the input and outputs, not the implementations

Invented in 1974 by Barbara Liskov

What we desire from an abstraction is a mechanism which permits the expression of relevant details and the suppression of irrelevant details. In the case of programming, the use which may be made of an abstraction is relevant; the way in which the abstraction is implemented is irrelevant. — Barbara Liskov

Source Article

Basic Definitions

Abstract Data Type (ADT)

- A definition for expected operations and behavior
- A mathematical description of a collection with a set of supported operations and how they should behave when called upon
- Describes what a collection does, not how it does it
- Can be expressed as an interface
- Examples: List, Map, Set

ADT = Data Type + Object Oriented Principles

Algorithm

- A series of precise instructions to produce to a specific outcome
- Examples: binary search, merge sort, recursive backtracking

Basic Definitions

A data structure is the implementation for an ADT .

• In an object oriented language such as Java, an ADT and its implementation together make up a class

• Each operation associated with the ADT is implemented by a member function or method .

• The variables that define the space required by a data item are referred to as

data members.

• An object is an instance of a class, something that is created and takes up storage during the execution of a computer program.

Features of ADT:

Abstract data types (ADTs) are a way of encapsulating data and operations on that data into a single unit. Some of the key features of ADTs include:

Abstraction: The user does not need to know the implementation of the data structure only essentials are provided.

Better Conceptualization: ADT gives us a better conceptualization of the real world.

Robust: The program is robust and has the ability to catch errors.

Encapsulation: ADTs hide the internal details of the data and provide a public interface for users to interact with the data. This allows for easier maintenance and modification of the data structure.

Data Abstraction: ADTs provide a level of abstraction from the implementation details of the data. Users only need to know the operations that can be performed on the data, not how those operations are implemented.

Data Structure Independence: ADTs can be implemented using different data structures, such as arrays or linked lists, without affecting the functionality of the ADT.

Information Hiding: ADTs can protect the integrity of the data by allowing access only to authorized users and operations. This helps prevent errors and misuse of the data.

Modularity: ADTs can be combined with other ADTs to form larger, more complex data structures. This allows for greater flexibility and modularity in programming.

Features of ADT:

Disadvantages:

Overhead: Implementing ADTs can add overhead in terms of memory and processing, which can affect performance.

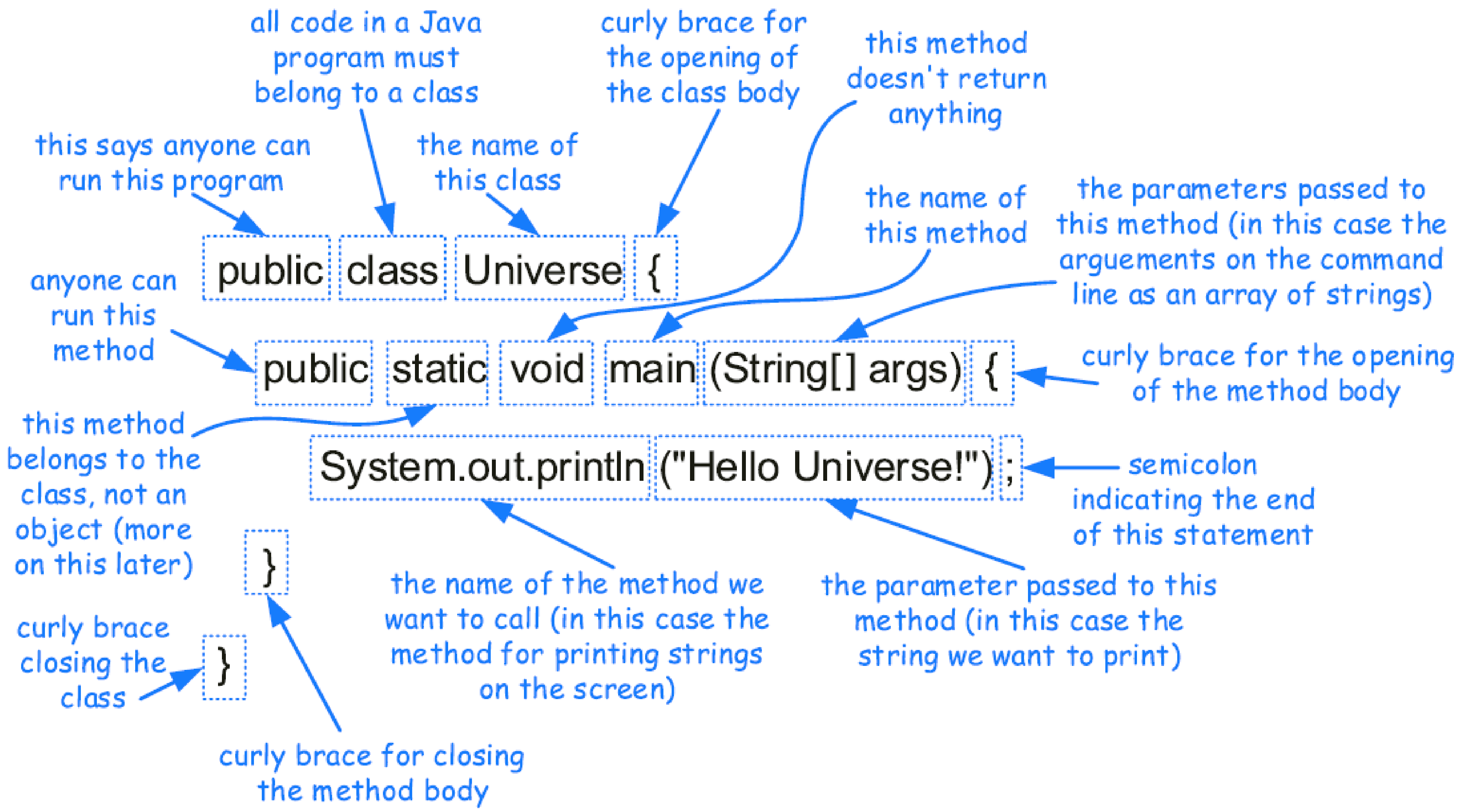
Complexity: ADTs can be complex to implement, especially for large and complex data structures.

Learning Curve: Using ADTs requires knowledge of their implementation and usage, which can take time and effort to learn.

Limited Flexibility: Some ADTs may be limited in their functionality or may not be suitable for all types of data structures.

Cost: Implementing ADTs may require additional resources and investment, which can increase the cost of development.

Review: A simple program in Java



```
public class Universe {
    public static void main(String[] args) {
        System.out.println("Hello Universe!");
    }
}
```

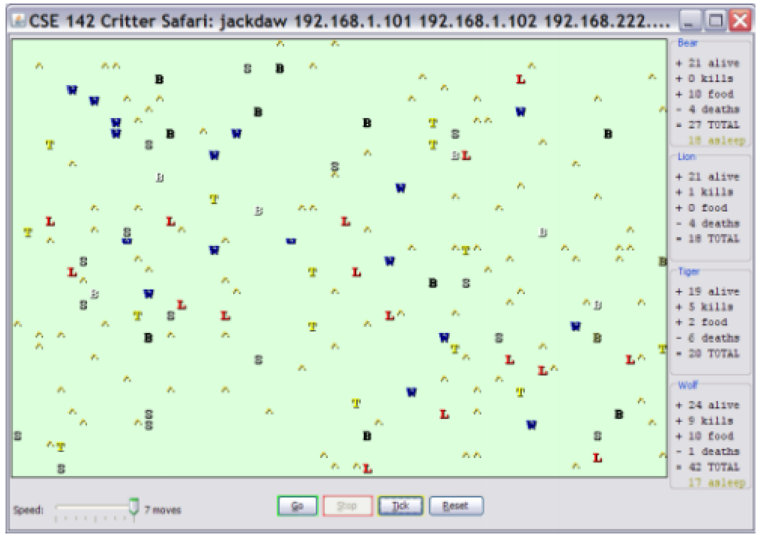
all code in a Java program must belong to a class
curly brace for the opening of the class body
this method doesn't return anything
this says anyone can run this program
the name of this class
the name of this method
the parameters passed to this method (in this case the arguements on the command line as an array of strings)
anyone can run this method
curly brace for the opening of the method body
this method belongs to the class, not an object (more on this later)
semicolon indicating the end of this statement
the name of the method we want to call (in this case the method for printing strings on the screen)
the parameter passed to this method (in this case the string we want to print)
curly brace closing the class
curly brace for closing the method body

Review: Clients vs Objects

CLIENT CLASSES

A class that is executable, in Java this means it contains a Main method

```
public static void main(String[] args)
```

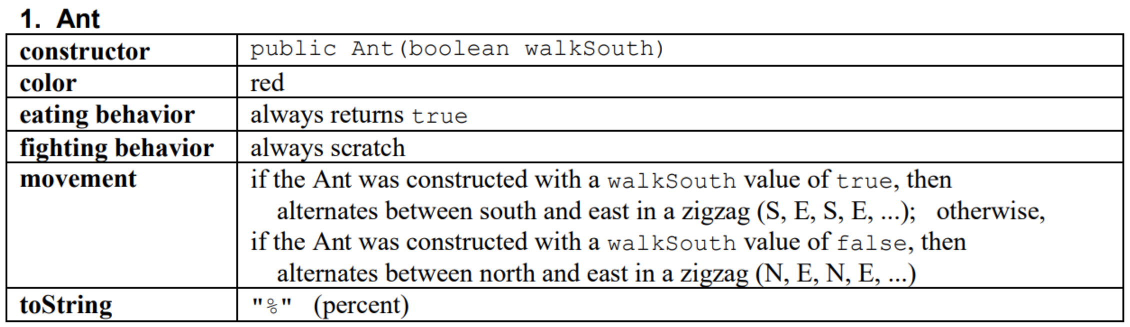

CSE 142 Critter Safari: jackdaw 192.168.1.101 192.168.1.102 192.168.222....
Speed:
7 moves
Go
Stop
Tick
Reset

```
Bear
+ 21 alive
+ 0 kills
+ 10 food
- 4 deaths
= 27 TOTAL
  18 asleep
Lion
+ 21 alive
+ 1 kills
+ 0 food
- 4 deaths
= 18 TOTAL
Tiger
+ 19 alive
+ 5 kills
+ 2 food
- 6 deaths
= 20 TOTAL
Wolf
+ 24 alive
+ 9 kills
+ 10 food
- 1 deaths
= 42 TOTAL
  17 asleep
```

OBJECT CLASSES

A coded structure that contains data and behavior

Start with the data you want to hold, organize the things you want to enable users to do with that data



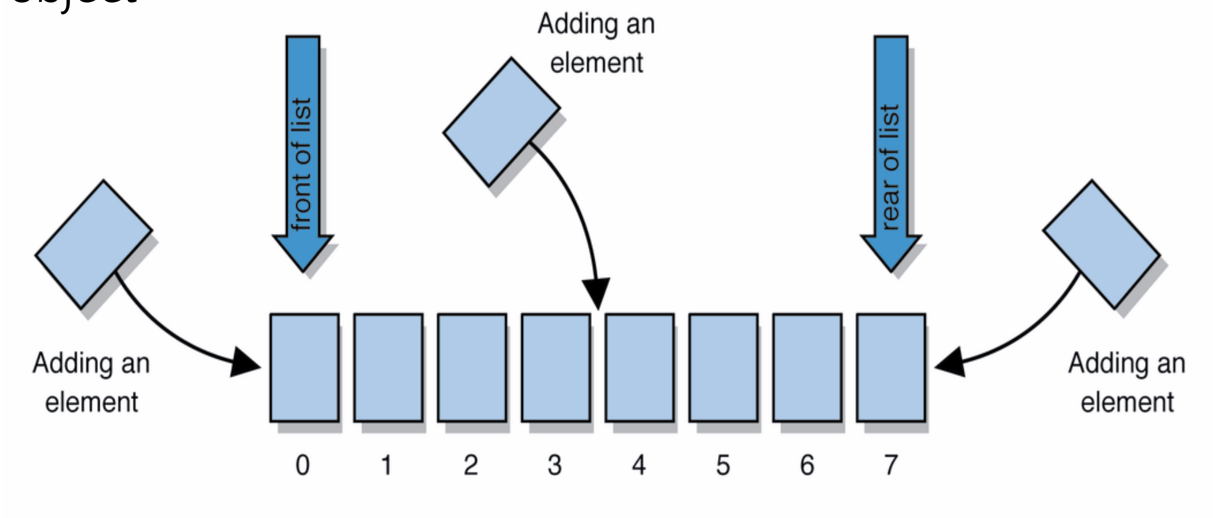
1. Ant

constructor	`public Ant(boolean walkSouth)`
color	red
eating behavior	always returns `true`
fighting behavior	always scratch
movement	if the Ant was constructed with a `walkSouth` value of `true`, then alternates between south and east in a zigzag (S, E, S, E, ...); otherwise, if the Ant was constructed with a `walkSouth` value of `false`, then alternates between north and east in a zigzag (N, E, N, E, ...)
toString	`"%"` (percent)

Abstract Data Types (ADT): List Example

Review: List - a collection storing an ordered sequence of elements

- each element is accessible by a 0-based index
- a list has a size (number of elements that have been added)
- elements can be added to the front, back, or elsewhere
- the ADT of a list can be implemented many ways through different Data Structures
    - in Java, a list can be represented as an ArrayList object



Review: Interfaces

interface: a construct in Java that defines a set of methods that a class promises to implement

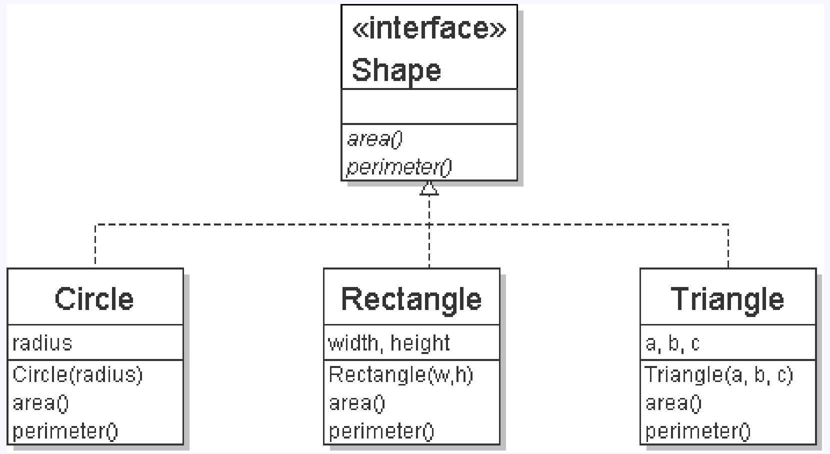
```
public interface name {
    public type name(type name, …, type name);
    public type name(type name, …, type name);
    …
}
```

- Interfaces give you an is-a relationship *without* code sharing.
  - A `Rectangle` object can be treated as a `Shape` but inherits no code.
- Analogous to non-programming idea of roles/certifications:
  - "I'm 'certified' as a Shape, because I implement the Shape interface. This assures you I know **how** to compute my area and perimeter."
  - "I'm 'certified' as a CPA accountant. This assures you I know **how** to do taxes, audits, and consulting."

Example

```
// Describes features common to all
// shapes.

public interface Shape {
        public double area();
        public double perimeter();
}
```



Review: Interfaces: List Example

interface: a construct in Java that defines a set of methods that a class promises to implement

In terms of ADTs, interfaces help us make sure that our implementations of that ADT are doing what they need to

For example, we could define an interface for the ADT `List<E>` and any class that implements it must have implementations for all of the defined methods

```
// Describes features common to all lists.

public interface List<E> {
    public E get(int index);
    public void set(E element, int index);
    public void append(E element);
    public E remove(int index);
    …

    // many more methods
}
```

note: this is not how the `List<E>` interface actually looks, as in reality it extends another interface

Review: Java Collections

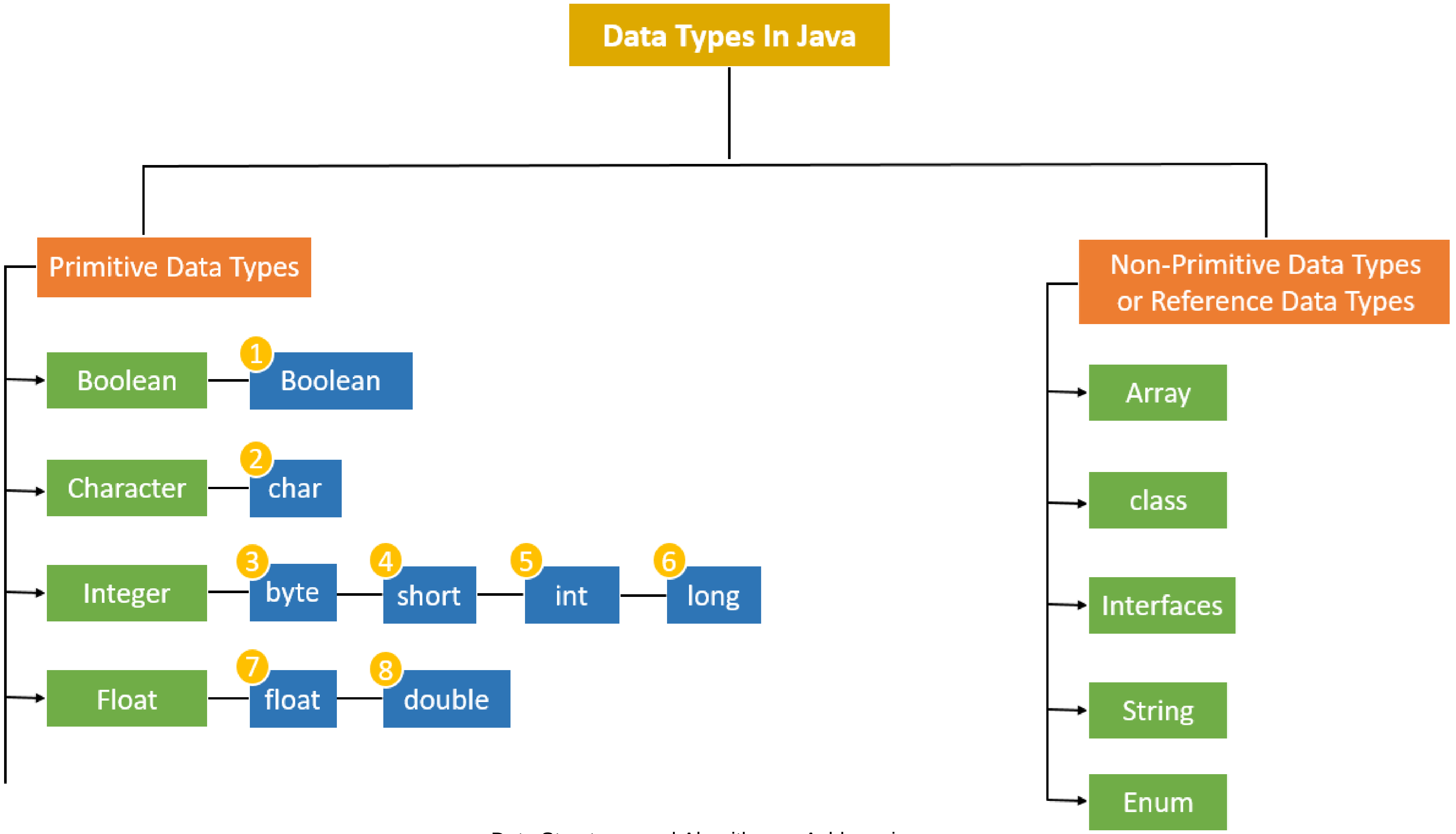
Java provides some implementations of ADTs for you!

ADTs	Data Structures
Lists	`List<Integer> a = new ArrayList<Integer>();`
Stacks	`Stack<Character> c = new Stack<Character>();`
Queues	`Queue<String> b = new LinkedList<String>();`
Maps	`Map<String, String> d = new TreeMap<String, String>();`

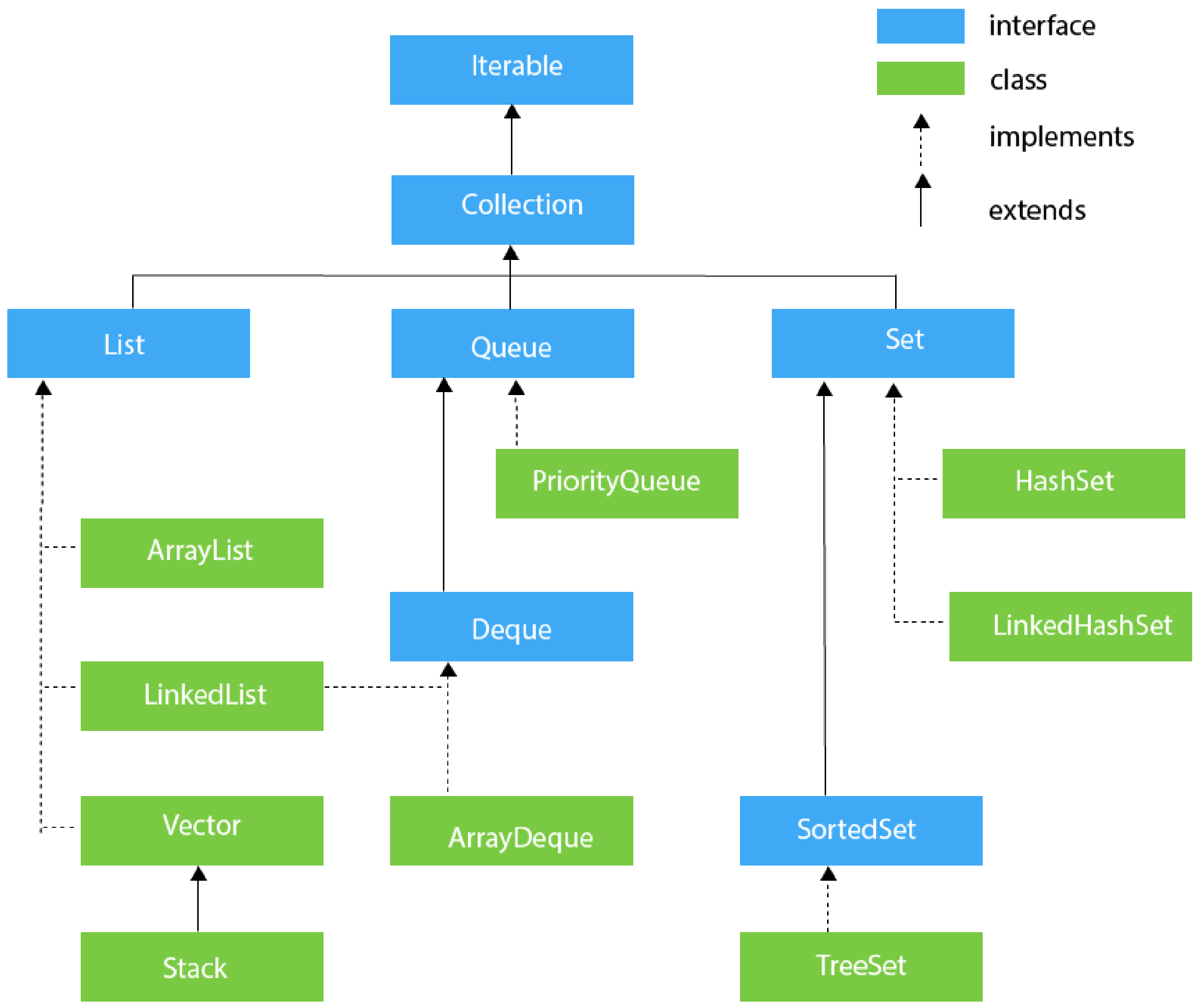
But some data structures you made from scratch… why?

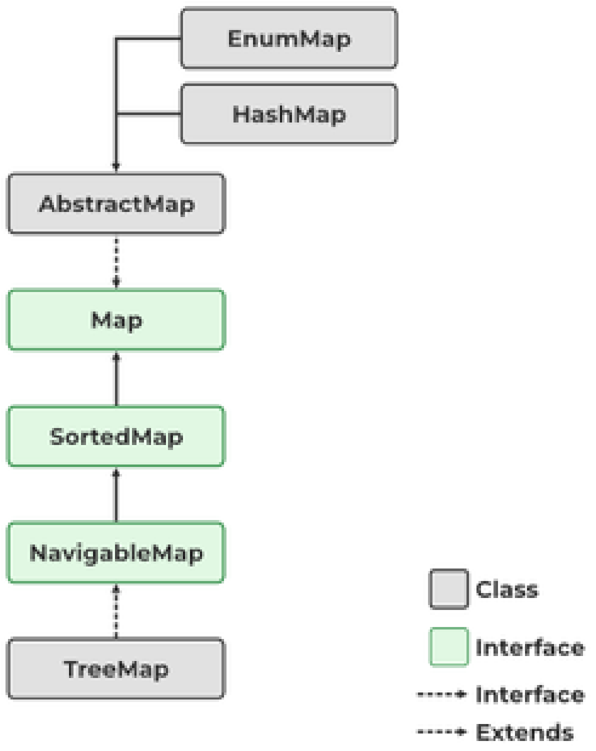
Linked Lists – LinkedIntList was a collection of ListNode
Binary Search Trees – SearchTree was a collection of SearchTreeNodes

Review: Java Datatypes



Review: Java Collections





ADTs we’ll discuss this quarter

- List: an ordered sequence of elements
- Set: an unordered collection of elements
- Map: a collection of “keys” and associated “values”
- Stack: a sequence of elements that can only go in or out from one end
- Queue: a sequence of elements that go in one end and exit the other
- Priority Queue: a sequence of elements that is ordered by “priority”
- Graph: a collection of points/vertices and edges between points
-

ADTs and Data Structures: Example

Abstract Data Type (ADT)

Mode of Transportation
Must be able to move
Must be able to be steered

Data Structures

Car	**Airplane**	**Bike**
Tires	Engines/wings	Wheels
Steering wheel	Control column	Handlebars

ADTs and Data Structures: List Example

List – Abstract Data Type (ADT)

```
// Describes features common to all lists.

public interface List<E> {
    public E get(int index);
    public void set(E element, int index);
    public void append(E element);
    public E remove(int index);
    …
}
```

ArrayIntList – Data Structure

```
public class ArrayIntList extends List<E>{
        private int[] list;
        private int size;

    public ArrayIntList(){
        //initialize fields
    }

    public int get(int index){
        return list[index];
    }

    public void set(E element, int index){
        list[index] = element;
    }
    …
}
```

Introductions

Syllabus

Basic Concept

Questions?