Data Structures and Algorithms

Chapter 1

Course Overview

The goal of this course is to learn fundamental components of computer programs.

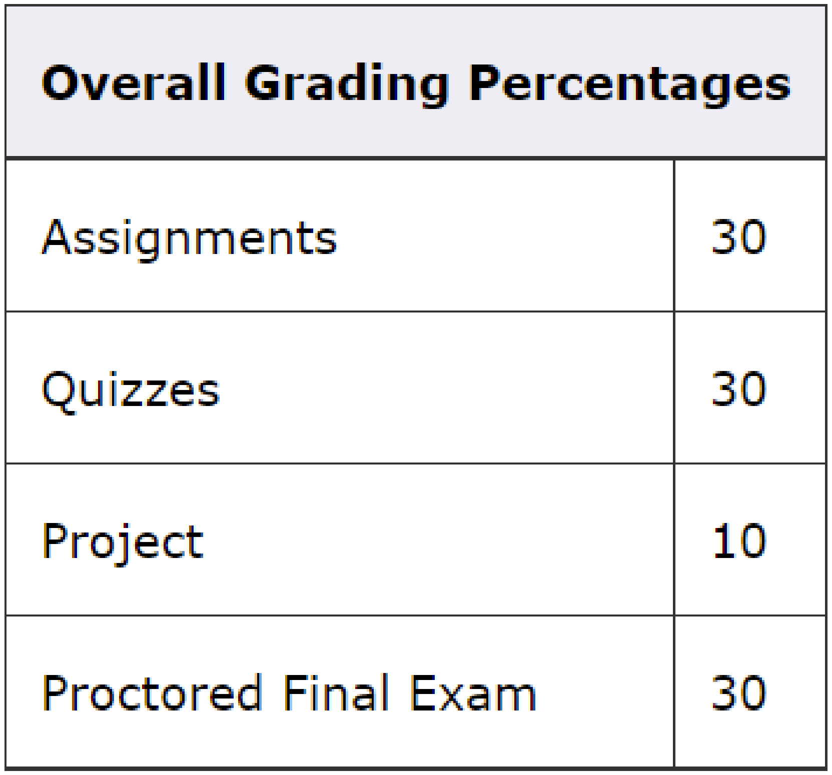
To use data structures to solve computational problems

And to implement data structures using a high-level programming language (Java)

Syllabus

- Module 1: Java Basics + Object-Oriented Design
- Module 2: Recursion, Stacks, Queues
- Module 3: Trees
- Module 4: Maps and Hash tables
- Module 5: Search Trees, Sorting, Greedy algorithms, Dynamic Programming
- Module 6: Computational Complexity

Grading



Overall Grading Percentages	
Assignments	30
Quizzes	30
Project	10
Proctored Final Exam	30

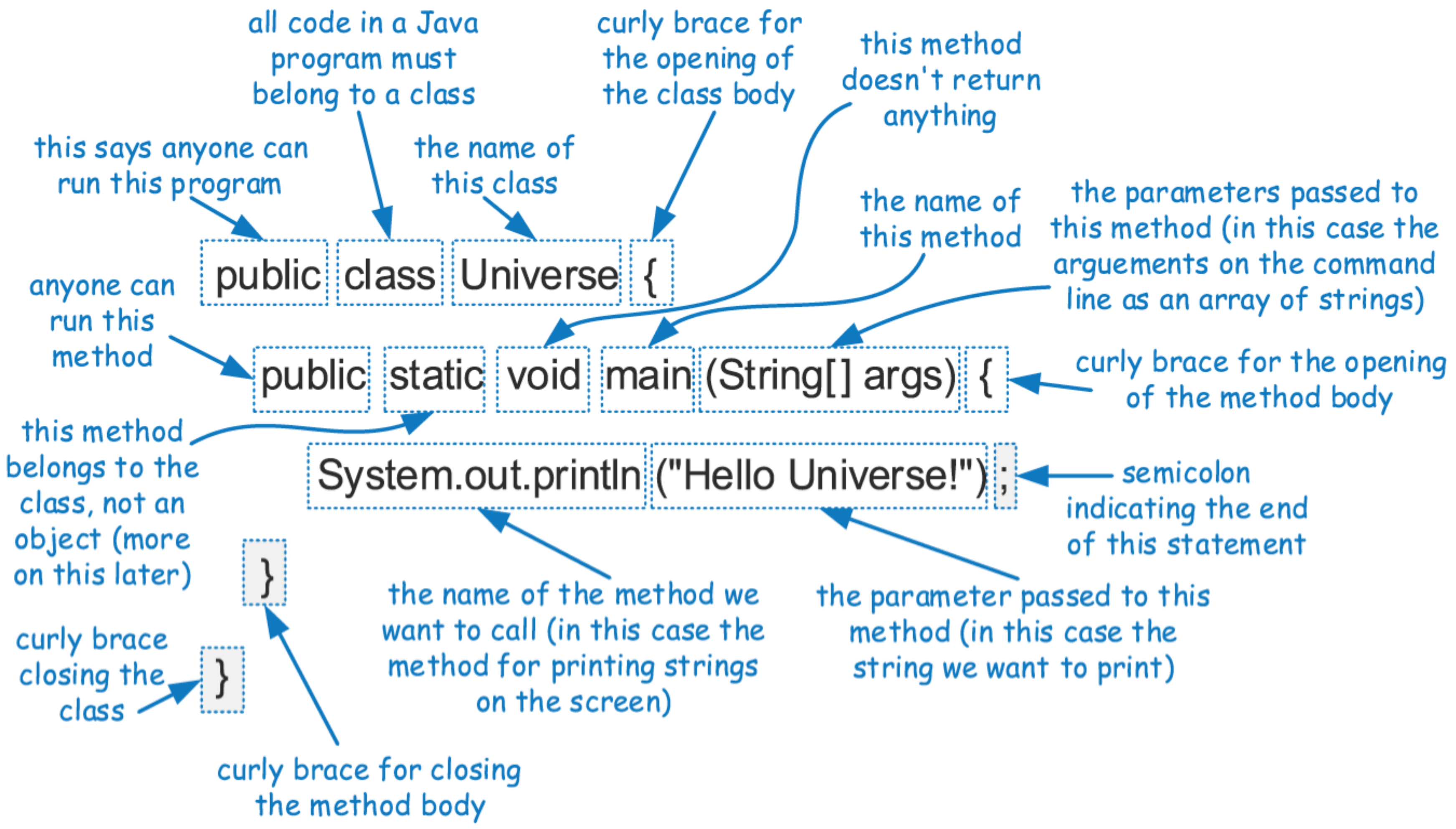
- **Assignments**: There is one assignment due each module (check the due date at the study guide). You submit the assignment in the "Assignments" area.
- **Quizzes**: There is one quiz due each module (check the due date at the study guide). You submit the assignment in the "Assessments" area.
- **Term Project**: There is a term project that is due at the end of the Module 6 (check the due date at the study guide). You submit the term project in the "Assignments" area.
- **Proctored Final Exam:** There will be a proctored Final Exam in this course (check the final exam period at the study guide). Detailed instructions regarding your proctored exam will be forthcoming from the Assessment Administrator. You will be responsible for scheduling your own appointment.

Lecture 1: Learning Objectives

- Understand basic Java data types
- Understand the basics of a Java program and keywords
- Understand control flow (loops, if-statements)
- Understand basics of using arrays

Java Basics

Sample Program



```
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
curly brace closing the class
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 for closing the method body

Java Basics

Components of a Java Program

- In Java, executable statements are placed in functions, known as **methods**, that belong to class definitions.
- The static method named **main** is the first method to be executed when running a Java program.
- Any set of statements between the braces "**{**" and "**}**" define a program block.

- Examples:
  - SampleProgram1.java
  - SampleProgram2.java

Java Basics
Primitive (or Base) Types

- Primitive types:
  - **byte**: 8-bit signed 2’s complement integer; from -128 to 127, inclusive
  - **short**: 16-bit signed 2’c complement integer; from -32768 to 32767, inclusive
  - **int**: 32-bit signed 2’s complement integer; from -2147483648 to 2147483647, inclusive
  - **long**: 64-bit signed 2’s complement integer;

    from -9223372036854775808 to 9223372036854775807, inclusive
  - **char**: 16-bit Unicode character;

    from '\u0000' to '\uffff' inclusive, that is, from 0 to 65535
  - **float**: single-precision, 32-bit floating point number (IEEE 754-1985)
  - **double**: double-precision, 64-bit floating point number (IEEE 754-1985)
  - **boolean**: true of false

Java Basics

Primitive (or Base) Types

- How to create primitive type variables:

```
boolean flag = true;
boolean verbose, debug;                   // two variables declared, but not yet initialized
char grade = 'A';
byte b = 12;
short s = 24;
int i, j, k = 257;                        // three variables declared; only k initialized
long l = 890L;                            // note the use of "L" here
float pi = 3.1416F;                       // note the use of "F" here
double e = 2.71828, a = 6.022e23;         // both variables are initialized
```

Java Basics

Casting

- Narrowing vs. widening type conversion

```
double x = 3.14
int a = (int)x; // narrowing conversion from
                // double to int
double y = a;   // widening conversion from int
                //to double
```

Java Basics

Reference Types

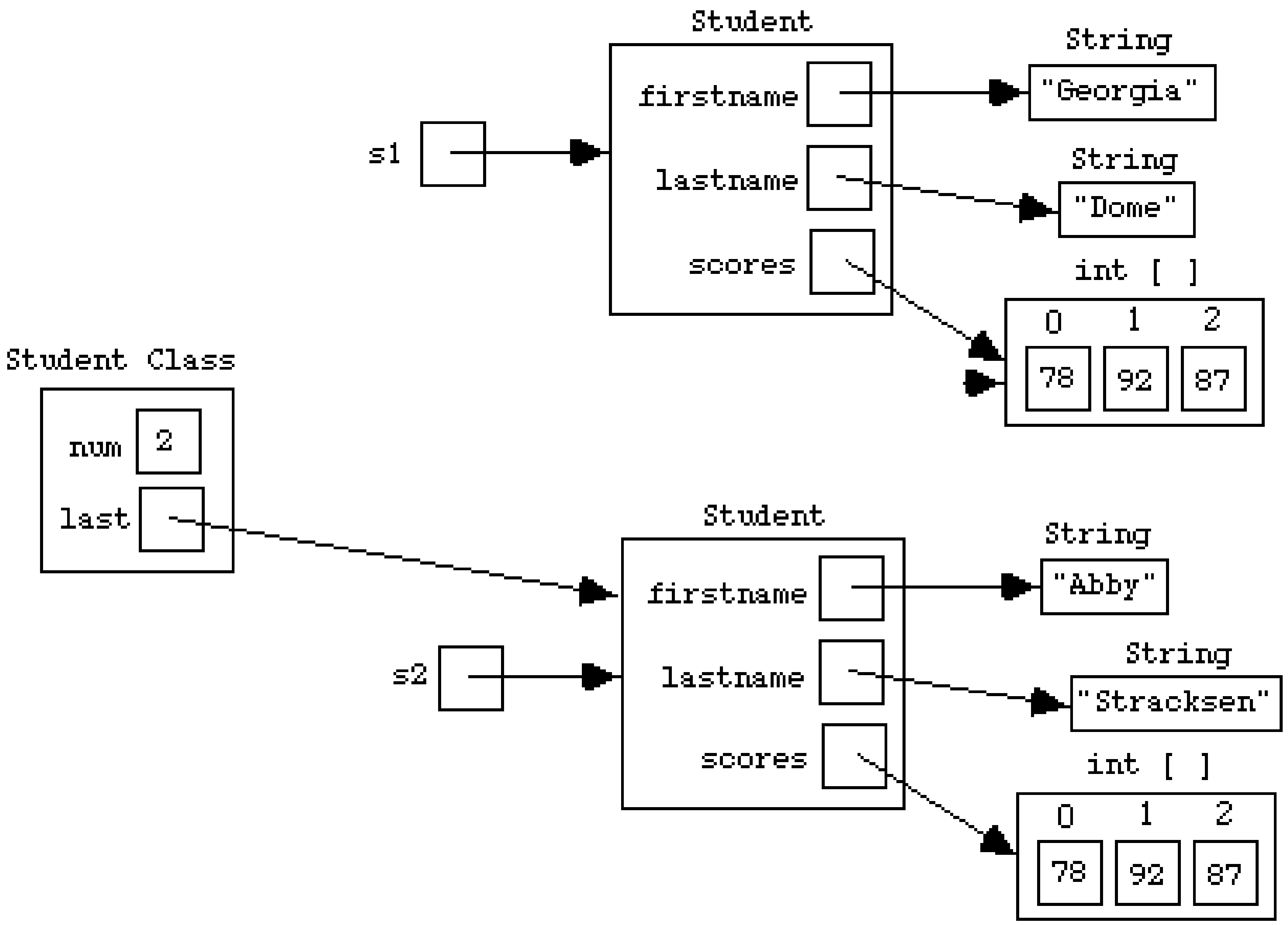
- Reference types: class types, interface types, array types.
- Values of a reference type: references to objects
- A reference variable stores the location (i.e., memory address) of an object.

- Example: Will see an example illustrating the difference between a primitive type and a reference type after we discuss creation of an object.

Classes and Objects

- In complex Java programs, primary actors are **objects**
- **Objects** are **instances** of a **class; a class** is a blueprint defining what an object stores/does
- **Methods** are blocks of code that can be called to perform actions
- **Instance variables** (or fields) hold the data associated with a single “instance” of an object

Classes and Objects



Classes and Objects

```
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
}
```

Java Basics

When a New Object is Created

- Memory is dynamically allocated.
- Instance variables are initialized .
- The *new* operator calls the constructor and returns the *reference* to the new object.
- The reference is assigned to an instance variable (a reference to the object).

```
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
```

Java Basics

Static Modifier

- Specified for variables or methods of a class.
- They belong to the class not to an instance of the class.

- Examples:
  - Car.java
  - PrimitiveReference.java
  - TestCar.java

Packages and Subclasses

- A **package** is a grouping of related classes
- If the package is named “cars”, all source code files must belong to a directory called “cars”

- Improves code organization and prevents naming conflicts

Java Basics

Access Control Modifier

- Also called *access level modifier* or *visibility modifier*.
- Declared for classes, variables, and methods.

Modifier	**Access Level**			
	Class	Package	Subclass	World
public	Y	Y	Y	Y
protected	Y	Y	Y	N
no modifier	Y	Y	N	N
private	Y	N	N	N

Java Basics

Control Flow

- if statements

```
if (booleanExpression)
    trueBody
else
    falseBody
```

Java Basics

Control Flow

- if statements

```
if (firstBooleanExpression)
    firstBody
else if (secondBooleanExpression)
    secondBody
else
    thirdBody
```

Java Basics

Control Flow

- **switch** statements

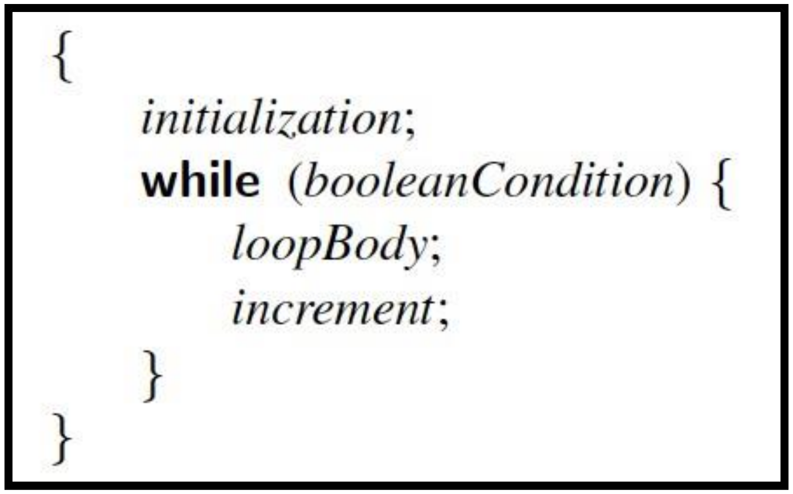
```
switch (var) {
    case value1:        // var == value1
        do something;
        break;
   case value2:        // var == value2
        do something;
        break;

   . . .

    default             // none of the above

        do something
}
```

Java Basics
Control Flow

- **for** loops

  for (initialization; booleanCondition; increment)
     loopBody

 Meaning:



```
{
    initialization;
    while (booleanCondition) {
        loopBody;
        increment;
    }
}
```

Java Basics
Control Flow

- **while** loops

```
while (booleanExpression)
    loopBody
```

- **do-while** loops

```
do
    loopBody
while (booleanExpression)
```

- Example: ControlFlowExamples.java

Java Basics

Arrays

- Declaration

```
int [ ] intArray; // array of integers
double [ ] doubleArray; // array of doubles
Char [ ] charArray; // array of characters
String [ ] stringArray; // array of strings
```

- Allocate memory, and initialize

```
intArray = new int [5];
IntArray[0] = 10;
IntArray[1] = 20;
IntArray[2] = 30;
IntArray[3] = 40;
IntArray[4] = 50;
```

Java Basics

Arrays

- Declare and allocate memory

  Int [ ] intArray = new int[10];

- Shortcut

  Int [ ] intArray = {10, 20, 30, 40, 50};

- Example: ArrayExample.java

Java Basics

Simple I/O

- Read from standard input and write to standard output example:
  - SimpleIOTest1.java
  - SimpleIOTest2.java
- Read from a text file and write to a text file:
  - SimpleIOTest3.java
  - There are other ways

References

- M.T. Goodrich, R. Tamassia, and M.H. Goldwasser, “Data Structures and Algorithms in Java,” Sixth Edition, Wiley, 2014.