FUNDAMENTALS

CS062

DATA STRUCTURES AND ADVANCED PROGRAMMING

2: Java Basics

Tom Yeh
he/him/his

Our team

Naomi Amuzie
she/her/hers

Carl Bell
he/him/his

Aidan Garton
he/him/his

George Johnson
he/him/his

Adeena Liang
they/them/their

Kacie Lee
she/her/hers

Max Rose
he/him/his

Ion Tsichrintzi
she/her/hers

Lecture 2: Java Basics

- Basics
- Methods
- Arrays
- Operators
- Control Flow

Some slides adopted from Algorithms, 4th Edition and Oracle tutorials

A possible implementation of a bicycle class in Java

```java
/**
 * Represents a bicycle
 * @author https://docs.oracle.com/javase/tutorial/java/concepts/class.html
*/
public class Bicycle {

    //instance variables
    private int cadence = 0;
    private int speed = 0;
    private int gear = 1;

    public void changeCadence(int newValue) {
         cadence = newValue;
    }

    public void changeGear(int newValue) {
         gear = newValue;
    }

    public void changeSpeed(int change) {
         speed = speed + change;
    }

    public int getCadence() {
         return cadence;
    }

    public void printGear() {
      System.out.println("Gear:" + gear);
    }

    public String toString() {
         return "cadence:" + cadence + " speed:" + speed + " gear:" + gear;
    }
}
```

- All code in a Java program must belong to a class.
- // comment within a line.
- /* multi-line comment.*/
- /**documentation comment (JavaDoc).*/
- The source code is saved in `.java` files.
- The name of the class should match the name of the source file e.g., `Bicycle.java.`
- Curly braces (`{` and `}`) are used to surround bodies of classes, methods, and loops.
- Statements end with a semicolon (`;`).
- Fields `cadence`, `speed`, `gear` represent the state of a bicycle object.
- Methods `changeCadence`, `changeGear`, etc. define how the object will interact with the world.
- `System.out.println` is Java's way of printing a string to the console.
- Override `toString` if you want to change how objects are printed (similar to Python).
- To run your code you will need a special method called `main` - there is no main in the `Bicycle` class.
- You can have a `main` method per class. Typically one of them will control the program and the rest will be used to test each class.

Bicycle Demo program

```
/**
 * Basic demonstration of how to work with bicycle objects
 * @author https://docs.oracle.com/javase/tutorial/java/concepts/class.html
 *
 */

public class BicycleDemo {
    public static void main(String[] args) {

        // Create two different Bicycle objects
        Bicycle bike1 = new Bicycle();
        Bicycle bike2 = new Bicycle();

        System.out.println(bike1);

        // Invoke methods on those objects
        bike1.changeCadence(50);
        bike1.changeSpeed(10);
        bike1.changeGear(2);
        bike1.printGear();
        System.out.println(bike1);

        bike2.changeCadence(50);
        bike2.changeSpeed(10);
        bike2.changeGear(2);
        bike2.changeCadence(40);
        bike2.changeSpeed(-10);
        bike2.changeGear(3);
        bike2.printGear();
        System.out.println(bike1);
        System.out.println(bike2);


    }
}
```

- In the `main` method, we instantiate two objects of type `Bicycle` with the `new` keyword, that is two new bicycles are being brought into this world.
- Object name + dot operator + method/variable to create a reference to an object's method/field
  - e.g., `bike1.changeCadence(50);`
- `Void` methods do not return anything.
  - `printGear` is `void`
- `System.out.println(someObject)` calls the `toString` method of the class `someObject` belongs to.

WHAT WILL THIS PROGRAM PRINT?

```
cadence:0 speed:0 gear:1
Gear:2
cadence:50 speed:10 gear:2
Gear:3
cadence:50 speed:10 gear:2
cadence:40 speed:0 gear:3
```

Access Modifiers

- `public` modifier - the field/method is accessible from all classes.
- `private` modifier - the field/method is accessible only within its own class.
- More that we will learn later…

Variables

- Containers for storing data values.
- Java is statically-typed: all variables must be declared along with their data type before they can be used.
  - e.g., `int cadence = 0;`
  - e.g., `String name;`
- Data types: primitives, classes, interfaces, and arrays.

Instance variables (non-static or member fields)

- Declared in a class but outside of any method.
- Each object has its own ***unique*** copy of the variable. E.g.,

```
public class Bicycle {

    private int cadence = 0;
    private int speed = 0;
    private int gear = 1;
}
```

- Invoked as `myObject.variableName`
- It's always a good idea to keep them `private`.

Static variables (class fields)

- Declared with the `static` modifier.
- All objects ***share the same copy***. E.g.,

```
public class Bicycle {

    public static int numberOfBicycles;
}
```

- Invoked as `ClassName.variableName`

USE SPARINGLY!

Local variables

- Declared ***within*** a method.
- Destroyed after the execution of the method.
- Can only be accessed within the method.
- No access modifier.
-
```
public int countToTen() {
    int counter = 0;
    //…
}
```

Naming Variables

- Variable names are case-sensitive.
- CamelCase.
- No white space.
- Start with small letter.
- Subsequent characters can be letters, digits, $, or _.
- Use full words that make sense.
- If name contains more than two words, capitalize the first letter of each subsequent word. e.g., `numberOfBicycles`.
- If your variable is a constant, capitalize everything. e.g., `PI`.

Identifier

- The name of a class, interface, method, or variable.
- Each category has its own naming conventions.

Reserved Words				
abstract	default	goto	package	synchronized
assert	do	if	private	this
boolean	double	implements	protected	throw
break	else	import	public	throws
byte	enum	instanceof	return	transient
case	extends	int	short	true
catch	false	interface	static	try
char	final	long	strictfp	void
class	finally	native	super	volatile
const	float	new	switch	while
continue	for	null		

Primitive Data Types

- Java supports 8 primitive data types.
- Primitives use a small amount of memory to represent a single item of data and support certain operations on its value.
- All data of same primitive data type use the same amount of memory.
- Cannot be used to instantiate type variables, that is no `new` keyword.
- Have corresponding object "wrapper" types:
  - `Integer`, `Double`, `Float`, `Boolean`, etc.
  - "Wrapper" types start with Capital letter: Integer vs int

Primitive Data Types

Type	Bits	Default	Example
byte	8	0	`byte b = 10;`
short	16	0	`short s = 2;`
int	32	0	`int i = 47;`
long	64	0L	`long l = 4747L;`
float	32	0.0f	`float f = 47.0f;`
double	64	0.0	`double d = 47.0;`
char	16	'\u0000'	`char c = 'a';`
boolean	1	false	`boolean fun = true;`

The compiler will assign default values to uninitialized instance and static fields.
If you do not initialize local variables you will run into a compile-time error!

The most important primitive data types to know

- `int` - for integers.
- `double` - for real numbers.
- `boolean` - for the set of values {`true`, `false`}.
- `char` - for alphanumeric characters and symbols.

- **STRINGS ARE NOT PRIMITIVES**
  - instead use class `String`.

Classes

- Main data types in Java.
  - e.g., `String`.
- Thousands more coming with Java by default.
- You can instantiate your own with the new keyword.
  - `Bicycle myBike = new Bicycle();`
- Contain fields (can be a primitive or class type) and methods.
- Respond to messages to communicate with the outside world by invoking methods.
- Reference default value is `null`.

A vocabulary refresher for variables

- Declaration: state the type of variable and its identifier. A variable can only be declared once. E.g., `int x;`
- Initialization: the first time a variable takes a value. E.g., `x = 3;`
  - Can be combined with declaration, e.g., `int y = 3;`
- Assignment: discarding the old value and replacing it with a new. E.g., `x = 2;`
- Static or instance variables are automatically initialized with default values, i.e. null for references to objects, 0 for int, false for boolean, etc.
- Local variables are not automatically initialized and your code won't compile if you have not initialized them and you are trying to use them. E.g.,

```
public void foo() {
    int x;
    System.out.println(x);
    //The local variable x might not have been initialized
}
```

Practice Time

Consider the following class:

```
public class IdentifyMyParts {
    public static int x = 7;
    public int y = 3;
}
```

a. What are the class/static variables?

b. What are the instance/member variables?

c. What are a and b?

d. What is the output from the following code:

```
IdentifyMyParts a = new IdentifyMyParts();
IdentifyMyParts b = new IdentifyMyParts();
a.y = 5;
b.y = 6;
a.x = 1;
b.x = 2;
System.out.println("a.y = " + a.y);
System.out.println("b.y = " + b.y);
System.out.println("a.x = " + a.x);
System.out.println("b.x = " + b.x);
System.out.println("IdentifyMyParts.x = " + IdentifyMyParts.x);
```

Answers

a. x

b. y

c. a.y = 5
   b.y = 6
   a.x = 2
   b.x = 2
   IdentifyMyParts.x = 2

https://docs.oracle.com/javase/tutorial/java/javaOO/QandE/creating-answers.html

Lecture 2: Java Basics

- Methods
- Arrays
- Operators
- Control Flow

Some slides adopted from Algorithms, 4th Edition and Oracle tutorials

Methods

- A collection of grouped statements that perform a logical operation and control the behavior of objects.
- By convention method names should be a verb (+ noun) in lowercase.
- Syntax: modifier returnType methodName(type parameter-name,…){…}
  - E.g., `public int getCadence(){…return cadence;}`
- Signature: method name and the number, type, and order of its parameters.
- Control goes back to the calling program as soon as a `return` statement is reached. If it does not return anything it is `void`.
- Can also be `static`, therefore shared by all instances of a class.
- Can be overloaded (same name, different parameters).

Constructors are invoked to create objects from class blueprints

- Constructor declarations look like method declarations but have the same name with the class and no return type

```
// the Bicycle class has one constructor
    public Bicycle(int startCadence, int startSpeed, int startGear) {
        gear = startGear;
        cadence = startCadence;
        speed = startSpeed;
    }
```

- To instantiate a new object use the new keyword

```
Bicycle myBike = new Bicycle(30, 0, 8);
```

- A class can have multiple constructors, including a no-argument constructor

```
// the Bicycle class could have a no-argument constructor
    public Bicycle() {
        gear = 1;
        cadence = 10;
        speed = 0;
    }

Bicycle yourBike = new Bicycle();
```

YOU DON'T HAVE TO PROVIDE A CONSTRUCTOR BUT IT'S ALWAYS A GOOD IDEA TO DO SO

`this` keyword

- Within an instance method or a constructor used to refer to current object.
- Can be used to call instance variables, methods and constructors. E.g.,

```
public class Point {
    private int x = 0;
    private int y = 0;

    //constructor
    public Point(int x, int y) {
        this.x = x;
        this.y = y;
    }
}
```

`this` keyword to invoke constructors

```
public class Rectangle {
    private int x, y;
    private int width, height;

    public Rectangle() {
        this(0, 0, 1, 1);
    }

    public Rectangle(int width, int height) {
        this(0, 0, width, height);
    }

    public Rectangle(int x, int y, int width, int height) {
        this.x = x;
        this.y = y;
        this.width = width;
        this.height = height;
    }
}
```

Parameters

- Variables passed in a method definition. You need to specify their type. E.g.,
- ```
  int countToNumber(int number) {
    //…
  }
  ```
- The arguments are the data you pass into the method's parameters. E.g., `countToNumber(3);`
  ```

Combination of instance/static variables/methods

- Instance methods can access instance variables and instance methods directly.
- Instance methods can access static variables and static methods directly.
- Static methods can access static variables and static methods directly.
- Static methods **cannot** access instance variables or instance methods directly—they must use an object reference.
  - E.g., "`Cannot make a static reference to the non-static field`" in `main` method
- Static methods cannot use the `this` keyword as there is no instance of an object for `this` to refer to.

Lecture 2: Java Basics

- Methods
- Arrays
- Operators
- Control Flow

Array: Our first data structure

- Container object that holds a sequence of a fixed number of values of the same type.
- The length of the array is established during its creation and stays fixed.
- Each item is called an element and each element is accessed by its index.
- If we have N elements the indices range from $0 \ldots N-1$
- Similar to a Python list, with some differences

Creating and initializing an array

1. Declare the array name and the type of its elements. E.g., `double[] a;`
2. Create the array. E.g., `a = new double[N];`
3. Initialize the array values. E.g.,

```
for (int i= 0; i<N; i++){
      a[i]=10.0;
}
```

- Default array initialization: We can combine all three steps into a single statement and all elements will take the default values (`0`, `false`, or `null` depending on type). E.g., `double[] a = new double[N];`
- Initializing declaration: List literal values between curly braces, separated by comma. E.g., `int[] b = {1,2,3};`

Using arrays

- Arrays have fixed size. We can access this size through its instance variable `length` (tsk, tsk, Java). E.g., `a.length`
- You can access or change an element using the `a[i]` notation.
- If you request an index that is either negative or larger than `length-1`, then you will get an ArrayIndexOutOfBoundsException.

Multidimensional arrays

```
/**
 * Illustration of multidimensional arrays
 *
 * @author https://docs.oracle.com/javase/tutorial/java/nutsandbolts/arrays.html
 *
 */
public class MultiDimArrayDemo {
    public static void main(String[] args) {
        String[][] names = {
            {"Mr. ", "Mrs. ", "Ms. "},
            {"King", "Park"}
        };
        // Mr. King
        System.out.println(names[0][0] + names[1][0]);
        // Mrs. Park
        System.out.println(names[0][1] + names[1][1]);
        // Ms. King
        System.out.println(names[0][2] + names[1][0]);
    }
}
```

Aliasing

- An array name refers to the whole array – if we assign one array name to another, then both refer to the same array.
- This can lead to aliasing problems.

```
int[] a = new int[N];
a[i] = 1234;
int[] b = a;
b[i] = 5678;   // a[i] is now 5678.
```

Practice Time:

1. The term "instance variable" is another name for ___.
2. The term "class variable" is another name for ___.
3. A local variable stores temporary state; it is declared inside a ___.
4. A variable declared within the opening and closing parentheses of a method signature is called a ____. The actual value passed is called an ___.
5. What are the eight primitive data types supported by the Java programming language?
6. Character strings are represented by the class ___.
7. An ___ is a container object that holds a fixed number of values of a single type.

Answers:

1. The term "instance variable" is another name for **non-static/member field**.
2. The term "class variable" is another name for **static field**.
3. A local variable stores temporary state; it is declared inside a **method**.
4. A variable declared within the opening and closing parentheses of a method is called a **parameter**. The actual value passed is called an argument.
5. What are the eight primitive data types supported by the Java programming language? **byte, short, int, long, float, double, boolean, char**
6. Character strings are represented by the class **java.lang.String**.
7. An **array** is a container object that holds a fixed number of values of a single type.

https://docs.oracle.com/javase/tutorial/java/nutsandbolts/QandE/answers_variables.html

Lecture 2: Java Basics

- Methods
- Arrays
- Operators
- Control Flow

Operator precedence

Operators	Precedence
postfix	`expr++ expr--`
unary	`+/++expr -/--expr !boolean`
multiplicative	`* / %`
additive	`+ -`
relational	`< > <= >= instanceOf`
equality	`== !=`
logical AND	`&&`
logical OR	`||`
assignment	`= += -= *= /=`

Assignment operator

- = assigns the value on its right to the operand on its left
  - e.g., `int cadence = 3;`

Arithmetic operators

```java
/**
 * Illustration of the arithmetic operators
 * @author https://docs.oracle.com/javase/tutorial/java/nutsandbolts/op1.html
 *
 */
public class ArithmeticDemo {

    public static void main (String[] args) {

        int result = 1 + 2;
        // result is now 3
        System.out.println("1 + 2 = " + result);
        int original_result = result;

        result = result - 1;
        // result is now 2
        System.out.println(original_result + " - 1 = " + result);
        original_result = result;

        result = result * 2;
        // result is now 4
        System.out.println(original_result + " * 2 = " + result);
        original_result = result;

        result = result / 2;
        // result is now 2
        System.out.println(original_result + " / 2 = " + result);
        original_result = result;

        result = result + 8;
        // result is now 10
        System.out.println(original_result + " + 8 = " + result);
        original_result = result;

        result = result % 7;
        // result is now 3
        System.out.println(original_result + " % 7 = " + result);
    }
}
```

Output:

```
1 + 2 = 3
3 - 1 = 2
2 * 2 = 4
4 / 2 = 2
2 + 8 = 10
10 % 7 = 3
```

Unary operators require only one operand

```
/**
 * Illustration of the unary operators
 * @author https://docs.oracle.com/javase/tutorial/java/nutsandbolts/op1.html
 *
 */
public class UnaryDemo {

    public static void main(String[] args) {

        int result = +1;
        // result is now 1
        System.out.println(result);

        result--;
        // result is now 0
        System.out.println(result);

        result++;
        // result is now 1
        System.out.println(result);

        result = -result;
        // result is now -1
        System.out.println(result);

        boolean success = false;
        // false
        System.out.println(success);
        // true
        System.out.println(!success);
    }
}
```

The ++/-- operators can be applied pre or post operand

```
/**
 * Illustration of the prefix/postfix unary operator
 * @author https://docs.oracle.com/javase/tutorial/java/nutsandbolts/op1.html
 *
 */
public class PrePostDemo {
    public static void main(String[] args){
        int i = 3;
        i++;
        // prints 4
        System.out.println(i);
        ++i;
        // prints 5
        System.out.println(i);
        // prints 6
        System.out.println(++i);
        // prints 6
        System.out.println(i++);
        // prints 7
        System.out.println(i);
    }
}
```

Equality/Relational operators

```
/**
 * Illustration of the equality/relational operators
 * @author https://docs.oracle.com/javase/tutorial/java/nutsandbolts/op1.html
 *
 */
public class ComparisonDemo {

    public static void main(String[] args){
        int value1 = 1;
        int value2 = 2;
        if(value1 == value2)
            System.out.println("value1 == value2");
        if(value1 != value2)
            System.out.println("value1 != value2");
        if(value1 > value2)
            System.out.println("value1 > value2");
        if(value1 < value2)
            System.out.println("value1 < value2");
        if(value1 <= value2)
            System.out.println("value1 <= value2");
    }
}
```

Conditional operators

```
/**
 * Illustration of the equality/relational operators
 * @author https://docs.oracle.com/javase/tutorial/java/nutsandbolts/op1.html
 *
 */
public class ConditionalDemo {

    public static void main(String[] args){
        int value1 = 1;
        int value2 = 2;
        if((value1 == 1) && (value2 == 2))
            System.out.println("value1 is 1 AND value2 is 2");
        if((value1 == 1) || (value2 == 1))
            System.out.println("value1 is 1 OR value2 is 1");
    }

}
```

A	B	A AND B	A OR B	NOT A
False	False	False	False	True
False	True	False	True	True
True	False	False	True	False
True	True	True	True	False

Practice Time

1. Consider the following code:
   ```
   arrayOfInts[j] > arrayOfInts[j+1]
   ```
   Which operators does the code contain?
2. Consider the following code snippet:
   ```
   int i = 10;
   int n = i++%5;
   ```
   a. What are the values of `i` and `n` after the code is executed?
   b. What are the final values of `i` and `n` if instead of using the postfix increment operator (`i++`), you use the prefix version (`++i`))?
3. To invert the value of a boolean, which operator would you use?
4. Which operator is used to compare two values, = or == ?

Answers:

1. >, +
2.
   a. `i` is 11, and `n` is 0
   b. `i` is 11, and `n` is 1.
3. The logical complement operator !
4. ==

https://docs.oracle.com/javase/tutorial/java/nutsandbolts/QandE/answers_operators.html

Lecture 2: Java Basics

- Methods
- Arrays
- Operators
- Control Flow

If-then statement

```
public void applyBrakes() {
    // the "if" clause: bicycle must be moving
    if (isMoving) { // condition must be inside parens
        // the "then" clause: decrease current speed
        currentSpeed--;
    }
}
```

If-then-else statement

```
/**
 * Illustration of the if then else control flow
 * @author https://docs.oracle.com/javase/tutorial/java/nutsandbolts/if.html
 *
 */
public class IfElseDemo {
    public static void main(String[] args) {

        int testscore = 76;
        char grade;

        if (testscore >= 90) {
            grade = 'A';
        } else if (testscore >= 80) {
            grade = 'B';
        } else if (testscore >= 70) {
            grade = 'C';
        } else if (testscore >= 60) {
            grade = 'D';
        } else {
            grade = 'F';
        }
        System.out.println("Grade = " + grade);
    }
}
```

ONCE A CONDITION IS SATISFIED, THE APPROPRIATE STATEMENTS ARE EXECUTED AND THE REMAINING CONDITIONS ARE NOT EVALUATED.

While statement

```
/**
 * Illustration of the if then else control flow
 * @author https://docs.oracle.com/javase/tutorial/java/nutsandbolts/while.html
 *
 */
public class WhileDemo {

    public static void main(String[] args){
        int count = 1;
        while (count < 11) {
            System.out.println("Count is: " + count);
            count++;
        }
    }
}
```

For statement

```
for (initialization; termination; increment) {
    statement(s)
}
```

```
/**
 * Illustration of the for loop
 * @author https://docs.oracle.com/javase/tutorial/java/nutsandbolts/for.html
 *
 */
public class ForDemo {
    public static void main(String[] args){
         for(int i=1; i<11; i++){
              System.out.println("Count is: " + i);
         }
    }
}
```

Enhanced for statement in most data structures

```
/**
 * Illustration of the enhanced for flow
 * @author https://docs.oracle.com/javase/tutorial/java/nutsandbolts/for.html
 *
 */
class EnhancedFor {
    public static void main(String[] args){
         int[] numbers =
             {1,2,3,4,5,6,7,8,9,10};
         for (int item : numbers) {
             System.out.println("Count is: " + item);
         }
    }
}
```

Break statement

- Use `break` to terminate a `for` or `while` loop.

```
/**
 * Illustration of the break branch
 *
 * @author https://docs.oracle.com/javase/tutorial/java/nutsandbolts/branch.html
 *
 */
public class BreakDemo {
      public static void main(String[] args) {

            int[] arrayOfInts = { 32, 87, 3, 589, 12, 1076, 2000, 8, 622, 127 };
            int searchfor = 12;

            int i;
            boolean foundIt = false;

            for (i = 0; i < arrayOfInts.length; i++) {
                  if (arrayOfInts[i] == searchfor) {
                        foundIt = true;
                        break;
                  }
            }

            if (foundIt) {
                  System.out.println("Found " + searchfor + " at index " + i);
            } else {
                  System.out.println(searchfor + " not in the array");
            }
      }
}
```

Continue statement

- Use `continue` to skip the current iteration of `for` or `while` loop.

```
 * Illustration of the continue branch
 *
 * @author https://docs.oracle.com/javase/tutorial/java/nutsandbolts/branch.html
 *
 */
public class ContinueDemo {
    public static void main(String[] args) {

        String searchMe = "peter piper picked a " + "peck of pickled peppers";
        int max = searchMe.length();
        int numPs = 0;

        for (int i = 0; i < max; i++) {
            // interested only in p's
            if (searchMe.charAt(i) != 'p')
                continue;                       // What happens if we used a break here?

            // process p's
            numPs++;
        }
        System.out.println("Found " + numPs + " p's in the string.");
    }
}
```

Return statement

- The `return` statement exits from the current method, and control flow returns to where the method was invoked.
- Can return a value, e.g., `return counter++;`
- Or not, e.g., `return;`

Lecture 2: Java Basics

- Methods
- Arrays
- Operators
- Control Flow

Readings:

- Oracle's guides:
  - Language Basics: https://docs.oracle.com/javase/tutorial/java/nutsandbolts/index.html
- Textbook:
  - Chapter 1.1 (Pages 8–35)
  - Chapter 1.2 (Pages 64–77, 84–88, 96–99)

Practice Problems:

- 1.1.1–1.1.5, 1.1.8–1.1.12, 1.2.4,1.2.8