University of Global Village (UGV), Barishal

Content of the Sessional Course
University Student (UGV) Format

Program: Bachelor of Science in Computer Science Engineering (CSE)

Course Code	--
Name of Course Title	**Java Programming (Level-6)**
Course Type	**Skill Course**
Level	**6th Semester**
Academic Session	**Winter 2025**
Name(s) of AcademicCourse teacher(s)	**Sohag Mollik, Lecturer in CSE.** **Mobile: 01304142395** **E-mail: sohag.cse.just@gmail.com**
Consultation Hours:	

Web Page design & development Lab Student (UGV) Format	
Course Code: --	**Credits: --**
Exam Hours: --	**CIE Marks: 30**
Course for 6th Semester, **Bachelor of Science in Computer Science Engineering (CSE)**	**SEE Marks: 20**

1.Course Learning Outcome (CLO) at the end of the course, the students will be able to-

CLO1: Fundamental of JAVA
CLO2: Data Structures and algorithms
CLO3: Advanced Java features
CLO4: Object-Oriented Programming (OOP)
CLO5: Real-World Applications and Optimization

2.Topics to be covered

Week	Topics	Teaching-Learning Strategy(s)	Class Hour	Practice Hour	Assessment Strategy(s)	Mapping with CLO
01	Demonstrate a strong understanding of Java fundamentals, including syntax, variables, data types.	Lecture, Live demonstration & Hands-on exercises.	5h	4h	Participation, Lab Performance	CLO 1
02	Explore control structures: if-else, switch statements, and loops (while, for).	Lecture, Interactive coding examples & Exercises.	5h	4h	Short quiz, Lab tasks	CLO 1
03	Arrays in Java: Declaration, initialization, and operations..	Lecture, Code-along & Problem-solving tasks.	5h	4h	Code reviews, Participation	CLO 1
04	Arrays in Java: Declaration, initialization, and operations.	Lecture, Hands-on exercises & Real-world examples.	5h	4h	Lab assignments, Class discussion.	CLO 2
05	Recursion in Java: Concepts, implementation, and use cases.	Lecture, Live demos, & Group exercises.	5h	5h	Participation, Mini-projects.	CLO 2
06	Recursion in Java: Concepts, implementation, and use cases..	Code-along, Practice tasks & Problem-solving.	5h	4h	Lab exercises, Quiz.	CLO 2
07	Java dates, math operations, and random number generation.	Interactive demonstration, Practical examples.	5h	5h	Lab assignments, Participation.	CLO 3

08	Exception handling in Java: Try-catch, finally, and custom exceptions.	Lecture, Hands-on challenges, Debugging tasks.	5h	4h	Debugging tasks, Short quiz.	CLO 3
09	Multithreading: Creating and managing threads, synchronization, and concurrency.	Code-along, Problem-solving & Case studies.	5h	4h	Lab exercises, Participation.	CLO 3
10	Multithreading: Creating and managing threads, synchronization, and concurrency.	Lecture, Data-driven tasks, Hands-on practice.	5h	4h	Lab assignments, Quiz.	CLO 3
11	Java algorithms: Sorting (basic and advanced) and searching techniques.	Lecture, Pattern-matching exercises.	5h	5h	Lab assignments, Quiz.	CLO 3
12	Lambda expressions and streams: Concepts, implementation, and practical use cases.	Lecture, Group exercises, and Coding practice.	5h	3h	Participation, Mini-projects.	CLO 3
13	Working with user input: Scanner class, file I/O, and data processing.	Practical debugging sessions, Code walkthrough.	5h	4h	Code debugging tasks.	CLO 3
14	Working with user input: Scanner class, file I/O, and data processing.	Lecture, Refactoring tasks, Group discussion	5h	5h	Lab reviews, Quiz.	CLO 4
15	Advanced OOP: Constructors, modifiers, encapsulation, and inheritance.	Lecture, Practical error-handling sessions.	5h	5h	Code review, Participation.	CLO 4
16	Advanced OOP: Constructors, modifiers, encapsulation, and inheritance.	Group project, Live guidance.	5h	5h	Project evaluation, Participation.	CLO 4
17	Final project development and deployment.	Project work, Mentoring	5h	2h	Project demonstration	CLO 5

3. Teaching-Learning Strategy:

- **Lecture:** Explain concepts with real-world examples and visual aids.
- **Live Demonstration:** Show step-by-step coding and debugging in real-time.
- **Interactive Coding Examples:** Engage students with challenges during class.
- **Hands-on Exercises:** Provide structured practice aligned with topics.
- **Code-along Sessions:** Guide students through practical coding implementations.
- **Problem-solving Tasks:** Assign real-world challenges to apply concepts.
- **Group Discussions:** Facilitate discussions on best practices and peer reviews.
- **Mini-projects:** Assign small, focused projects incorporating multiple concepts.
- **Debugging Sessions:** Teach error identification and resolution with tools.
- **Final Project Work:** Mentor students in developing a complete web application.

4.Assessment Strategy:

- **Lab Performance:** 30% (Lab participation, hands-on exercises, and weekly assessments)
- **Quizzes and Short Tests:** 20% (Regular quizzes on theoretical concepts)
- **Assignments and Reports:** 20% (Assignments related to data management, cloud integration, and security)
- **Project Evaluation:** 30% (Progress, final project implementation, and presentation)

5.Instructional Materials and References: Textbooks:

1."Head First Java" by Kathy Sierra and Bert Bates
2."Effective Java" by Joshua Bloch

Additional References:

Follow w3school Java & others website.

WEEK 1

Page
7-14

Java

- A programming language employs a set of rules that dictate how the words and symbols can be put together to form valid *program statements*
- The Java programming language was created by Sun Microsystems.
- It was introduced in 1995 and it's popularity has grown quickly

Program Development

- The mechanics of developing a program include several activities
  - writing the program in a specific programming language (such as Java)
  - translating the program into a form that the computer can execute
  - investigating and fixing various types of errors that can occur
- Software tools can be used to help with all parts of this process

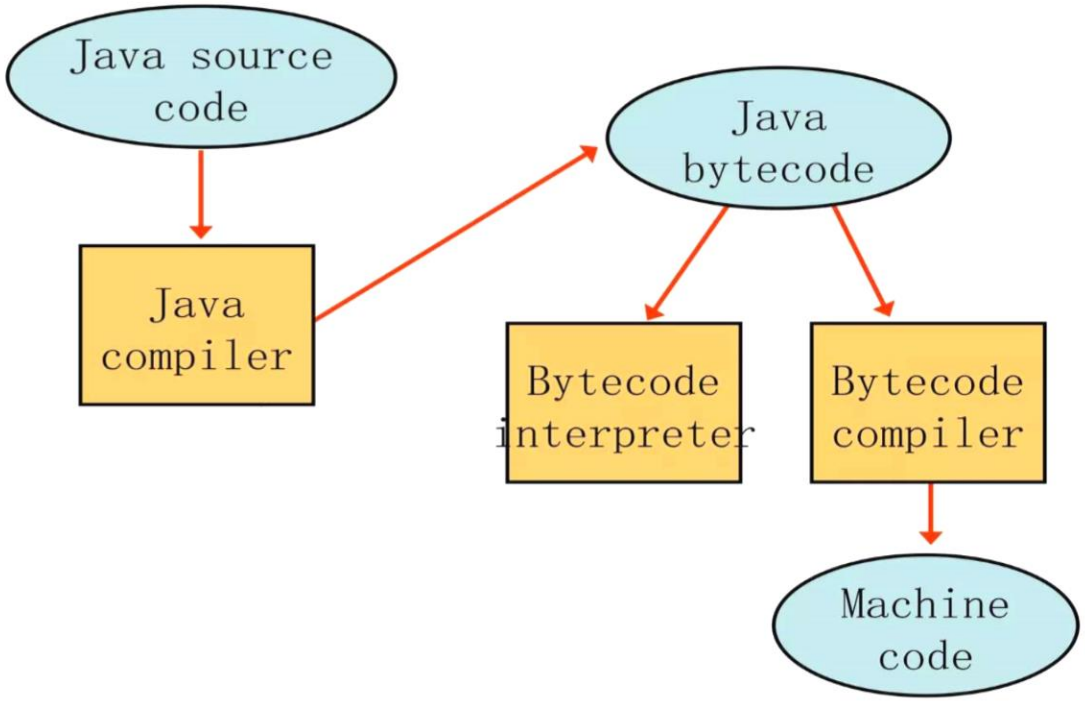
Program Development

- Each type of CPU executes only a particular *machine language*
- A program must be translated into machine language before it can be executed
- A *compiler* is a software tool which translates *source code* into a specific target language
- Often, that target language is the machine language for a particular CPU type
- The Java approach is somewhat different

Java Translation

- The Java compiler translates Java source code into a special representation called *bytecode*
- Java bytecode is not the machine language for any traditional CPU
- Another software tool, called an *interpreter*, translates bytecode into machine language and executes it
- Therefore the Java compiler is not tied to any particular machine
- Java is considered to be *architecture-neutral*

Java Translation



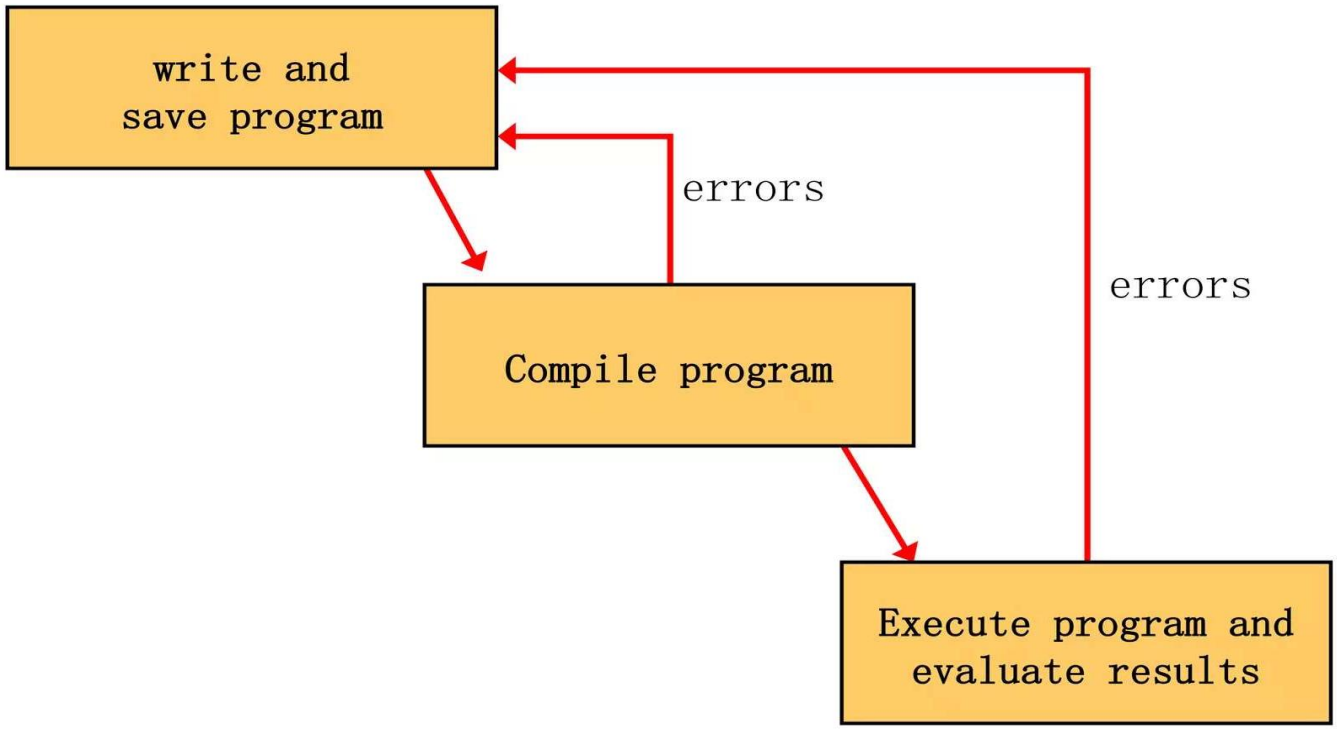
Development Environments

- There are many programs that support the development of Java software, including:
  - Sun Java Development Kit (JDK)
  - Sun NetBeans
  - IBM Eclipse
  - Borland JBuilder
  - MetroWerks CodeWarrior
  - BlueJ
  - jGRASP
- Though the details of these environments differ, the basic compilation and execution process is essentially the same

Java Program Structure

- In the Java programming language:
  - A program is made up of one or more *classes*
  - A class contains one or more *methods*
  - A method contains program *statements*
- These terms will be explored in detail throughout the course
- A Java application always contains a method called main

Basic Program Development



WEEK 2

Page
16-21

First.java

```
public class First
{
   //-----------------------------------------------
   //  Prints  Einstein quote.
   //-----------------------------------------------
   public static void main (String[] args)
   {
      System.out.println ("Albert Einstein Quote:");

      System.out.println (" Education is not the learning
   of facts, but the training of the mind to think.");
   }
}
```

Java Program Structure

```
// comments about the class

public class MyProgram
{
    // comments about the method
    public static void main (String[] args)
    {

    }

}
```

method header

method body

Comments

- Comments in a program are called *inline documentation*
- They should be included to explain the purpose of the program and describe processing steps
- They do not affect how a program works
- Java comments can take three forms:

```
// this comment runs to the end of the line

/*  this comment runs to the terminating
    symbol, even across line breaks       */

/** this is a javadoc comment   */
```

Identifiers

- *Identifiers* is a name given to a piece of data, method, etc.
- Identifiers give names to: classes, methods, variables, constants ….
- An identifier can be made up of letters, digits, the underscore character (_), and the dollar sign($)
- Identifiers cannot begin with a digit
- Java is *case sensitive* - e.g Total, total, and TOTAL are different identifiers

Identifiers

- Conventions for naming in Java:
  - classes: capitalize each word (ClassName)
  - methods: capitalize each word after the first (methodName)(variable names follow the same convention)
  - constants: all caps, words separated by _ (CONSTANT_NAME)

Identifiers

- susan
- second_place
- _myName
- TheCure
- ANSWER_IS_42
- $variable

legal

- me+u
- 49er
- question?
- side-swipe
- hi there
- jim's
- 2%milk
- suzy@yahoo.com

illegal

Guys can you explain why each of the above identifiers is not legal?

WEEK 3

Page
22-29

Reserved Words

- The Java reserved words:

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

White Space

- Spaces, blank lines, and tabs are called *white space*
- White space is used to separate words and symbols in a program
- Extra white space is ignored by java compilers

  int a; and int a;
- Programs should be formatted to enhance readability, using consistent indentation

Second. java

```
public class Second
{
public static void main(String[]args)
{
System.out.print("A quote by Abraham Lincoln:");
System.out.        println("Whatever you are, be a
   good one.");
}
}
```

Syntax and Semantics

- The syntax rules of a language define how we can put together symbols, reserved words, and identifiers to make a valid program
- The semantics of a program statement define what that statement means (its purpose or role in a program)
- A program that is syntactically correct is not necessarily logically (semantically) correct
- A program will always do what we tell it to do, not what we meant to tell it to do

Errors

- A program can have three types of errors
- The compiler will find syntax errors and other basic problems (*compile-time errors*)
  - If compile-time errors exist, an executable version of the program is not created
- A problem can occur during program execution, such as trying to divide by zero, which causes a program to terminate abnormally (*run-time errors*)
- A program may run, but produce incorrect results, perhaps using an incorrect formula (*logical errors*)

Data Types

- Primitive Data Types
  - Byte, short, int, long, float, double, Boolean, char

```
//dataType identifier;
int x;
int y = 10;
int z, zz;
double a = 12.0;
boolean done = false, prime = true;
char mi = 'D';
```

  - stick with int for integers, double for real numbers

Java Primitive Data Types

Data Type	Characteristics	Range
byte	8 bit signed integer	-128 to 127
short	16 bit signed integer	-32768 to 32767
int	32 bit signed integer	-2,147,483,648 to 2,147,483,647
long	64 bit signed integer	-9,223,372,036,854,775,808 to-9,223,372,036,854,775,807
float	32 bit floating point number	± 1.4E-45 to ± 3.4028235E+38
double	64 bit floating point number	± 4.9E-324 to ± 1.7976931348623157E+308
boolean	true or false	NA, note Java booleans cannot be converted to or from other types
char	16 bit, Unicode	Unicode character, \u0000 to \uFFFF Can mix with integer types

WEEK 04-06

Page
30-47

What are Classes and Objects?

- Class is synonymous with data type
- Object is like a variable
  - The data type of the Object is some Class
  - referred to as an *instance of a Class*
- Classes contain:
  - the implementation details of the data type
  - and the interface for programmers who just want to use the data type
- Objects are complex variables
  - usually multiple pieces of internal data
  - various behaviors carried out via *methods*

- Declaration – DataType identifier

```
Rectangle r1;
```

Built in Classes

- Java has a large built in library of classes with lots of useful methods

- System
- Arrays
- Scanner
- File
- Object
- Random
- String
- Math
- Integer, Character, Double

Casting

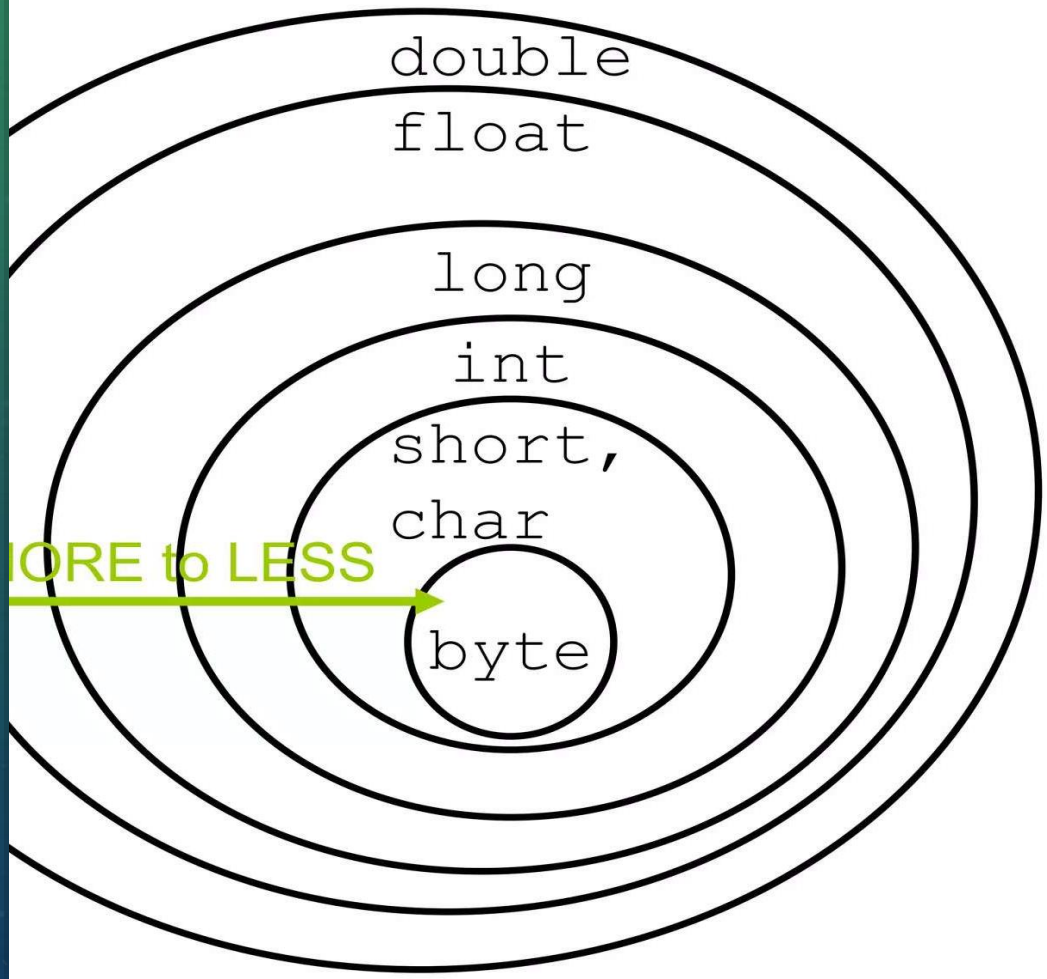
- Casting is the temporary conversion of a variable from its original data type to some other data type.
  - Like being cast for a part in a play or movie
- With primitive data types if a cast is necessary from a less inclusive data type to a more inclusive data type it is done automatically.

```
int x = 5;
double c = x / 2;
```

- if a cast is necessary from a more inclusive to a less inclusive data type the class must be done explicitly by the programmer

```
double a = 3.5, b = 2.7;
int y = (int) a / (int) b;
```

Primitive Casting



Outer ring is most inclusive data type. Inner ring is least inclusive.

In expressions variables and sub expressions of less inclusive data types are automatically cast to more inclusive.

If trying to place expression that is more inclusive into variable that is les inclusive, explicit must be performed.

Variables

- A *variable* is a name for a location in memory
- A variable must be *declared* by specifying the variable's name and the type of information that it will hold


```
int total;

int count, temp, result;
```

Multiple variables can be created in one declaration

Variable Initialization

- A variable can be given an initial value in the declaration

```
int sum = 0;
int base = 32, max = 149;
```

- When a variable is referenced in a program, its current value is used

Three. java

```
public class Three
{
   //-----------------------------------------------
   //Prints the number of keys on a piano.
   //-----------------------------------------------
   public static void main (String[] args)
   {
      int keys = 88;

      System.out.println ("A piano has " + keys + " keys.");
   }
}
```

Assignment

- An *assignment statement* changes the value of a variable
- The assignment operator is the = sign

```
total = 55;
```

- The expression on the right is evaluated and the result is stored in the variable on the left
- The value that was in total is overwritten
- You can only assign a value to a variable that is consistent with the variable's declared type

Geometry.java

```
public class Geometry
{
   public static void main (String[] args)
        {
        int sides = 7;  // declaration with initialization
        System.out.println ("A heptagon has " + sides + " sides.");

        sides = 10;  // assignment statement
        System.out.println ("A decagon has " + sides + " sides.");

        sides = 12;
        System.out.println ("A dodecagon has " + sides + "
   sides.");
   }
}
```

Constants

- A constant is an identifier that is similar to a variable except that it holds the same value during its entire existence
- As the name implies, it is constant, not variable
- The compiler will issue an error if you try to change the value of a constant
- In Java, we use the final modifier to declare a constant

```
final int MIN_HEIGHT = 69;
```

Operators

- Basic Assignment: =
- Arithmetic Operators: +, -, *, /, %(remainder)
  - integer, floating point, and mixed arithmetic and expressions
- Assignment Operators: +=, -=, *=, /=, %=
- increment and decrement operators: ++, --
- A Java expression consists of variables, operators, literals, and method calls.
- Here, score = 90 is an expression that returns an int .
- Consider another example, Double a = 2.2, b = 3.4, result; result = a + b - 3.4;
- Here, a + b - 3.4 is an expression

String Concatenation

- The + operator is also used for arithmetic addition
- The function that it performs depends on the type of the information on which it operates
- If both operands are strings, or if one is a string and one is a number, it performs string concatenation
- If both operands are numeric, it adds them

The println Method

- In the second program from the previous, we invoked the println() method to print a character string
- The System.out object represents a destination (the monitor screen) to which we can send output

```
System.out.println ("Whatever you are, be a good one.");
```

The print Method

- The **System.out** object provides another service as well
- The print method is similar to the **println** method, except that it does not advance to the next line
- Therefore anything printed after a print statement will appear on the same line

Countdown. java

```
public class Countdown
{
   //-----------------------------------------------------
   //  Prints two lines of output representing a rocket c
   //-----------------------------------------------------
   public static void main (String[] args)
   {
      System.out.print ("Three... ");
      System.out.print ("Two... ");
      System.out.print ("One... ");
      System.out.print ("Zero... ");

      System.out.println ("Liftoff!");// appears on first
   output line

      System.out.println ("Houston, we have a problem.");
   }
}
```

Control Structures

- linear flow of control
  - statements executed in consecutive order
- Decision making with if - else statements

```
if(boolean-expression)
    statement;
if(boolean-expression)
{   statement1;
    statement2;
    statement3;
}
```

A single statement could be replaced by a statement block, braces with 0 or more statements inside

Boolean Expressions

- boolean expressions evaluate to true or false
- Relational Operators: >, >=, <, <=, ==, !=
- Logical Operators: &&, ||, !
  - && and || cause short circuit evaluation
  - if the first part of p && q is false then q is not evaluated
  - if the first part of p || q is true then q is not evaluated

```
//example
if( x <= X_LIMIT && y <= Y_LIMIT)
    //do something
```

WEEK 07-08

Page
49-60

More Flow of Control

- if-else:

```
if(boolean-expression)
    statement1;
else
    statement2;
```

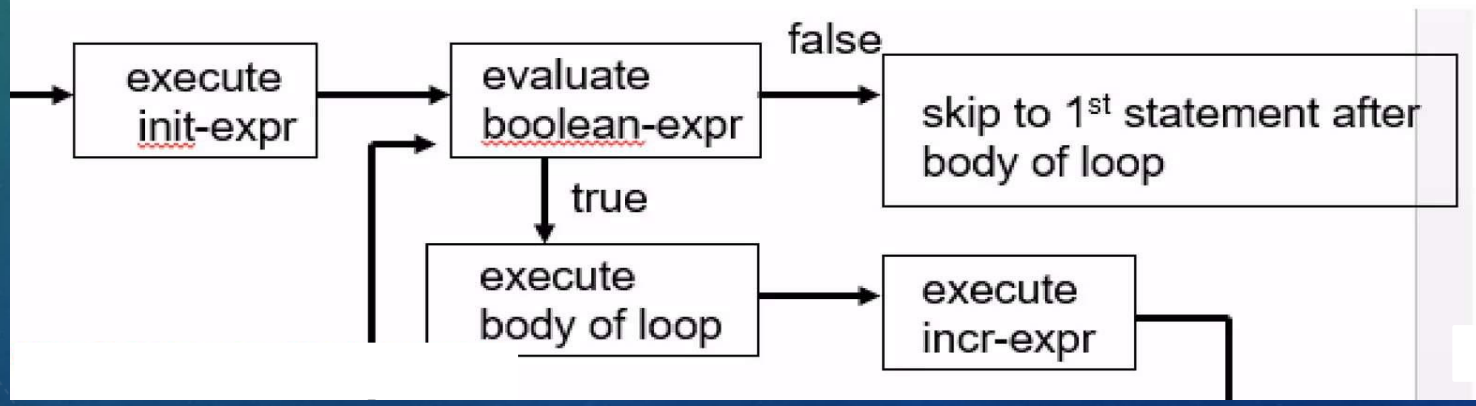
- multiway selection:

```
if(boolean-expression1)
    statement1;
else if(boolean-expression2)
    statement2;
else
    statement3;
```

- individual statements could be replaced by a statement block, a set of braces with 0 or more statements

for Loops

- for loops
  for(init-expr;boolean-expr;incr-expr)
      statement;
- init-expr and incr-expr can be more zero or more expressions or statements separated by commas
- statement could be replaced by a statement block



while loops

- while loops

```
while(boolean-expression)
    statement; //or statement block
```

- do-while loop part of language

```
do
    statement;
while(boolean-expression);
```

The switch Statement

General form of a switch statement:

```
switch (SwitchExpression) {
      case CaseExpression1:
            //One or more statements
            break;
      case CaseExpression2:
            //One or more statements
            break;
      default:
            //One or more statements
}
```

The switch Statement

- switch - keyword that begins a switch statement
- *SwitchExpression* - a variable or expression that has to be either char, byte, short, or int.
- case - keyword that begins a case statement (there can be any number of case statements)
- *CaseExpression* - a literal or final variable that is of the same type as *SwitchExpression* .
- The CaseExpression of each case statement must be unique.
- The default section is optional.
- Without the break; at the end of the statements associated with a case statement, the program "falls through" to the next case statement's statements, and executes them.

The switch Statement

```
if (x == 1)
        y = 4;
if else (x == 2)
        y = 9;
else
        y = 22;
```

Is the same as…

```
switch (x) {
        case 1:
                y = 4;
                break;
        case 2:
                y = 9;
                break;
        default:
                y = 22;
        }
```

Print the F.F outputs

Program-1

```
public class Student {
public static void main(String[] args) {
int x = 12;
int y = 12;
if(x+y > 20) {
System.out.println("x + y is greater than 20");

} } }
```

Program 2

```
public class Student {
public static void main(String[] args) {
int x = 10;
int y = 12;
if(x+y < 10) {
System.out.println("x + y is less than 10");
}   else {
System.out.println( "welcome 3rd year group_1");
}
}
```

Program 3

```
public class Student {
public static void main(String[] args) {
String city = ” Tepi” ;
if(city == ” Mizan” ) {
System.out.println( “city is Mizan” );
}else if (city == ” Addis Ababa” ) {
System.out.println( “city is  Addis Ababa  “);
}else if(city ==  “Bahirdar” ) {
System.out.println( “ Bahirdar  “);
}else {
System.out.println(city);
}  } }
```

Program 4

```
public class Calculation {
public static void main(String[] args) {
int sum = 0;
for(int j = 1; j<=10; j++) {
sum = sum + j;
}
System.out.println("The sum of first 10 natural nu
mbers is " + sum);
}
}
```

Program 5

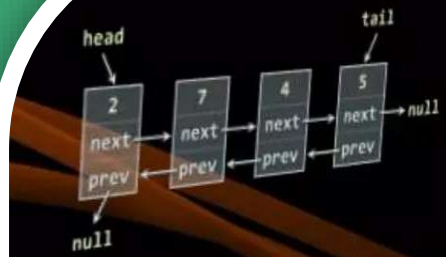
```
public class Calculation {
public static void main(String[] args) {
int i = 0;
System.out.println( “Printing the list of first 10
 even numbers \n” );
while(i<=10) {
System.out.println(i);
i = i + 2;
}  }  }
```

Program-7

```
int k =  2;
switch (k){
 case 1:
 System.out.println("A");
 case 2:case 3:
 System.out.println("B");
 break;
 case 4:
 System.out.println("C");
 default:
 System.out.println("D"); }
```

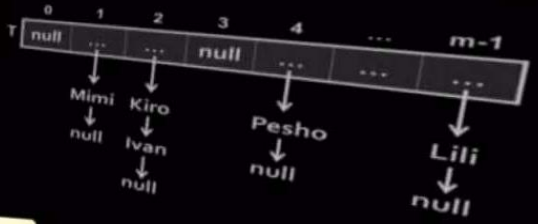
WEEK 9-12

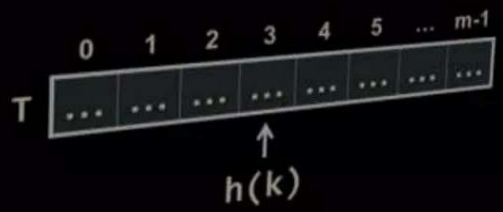
Page
62-93



Java Collections Basics

Arrays, Lists, Strings, Sets, Maps





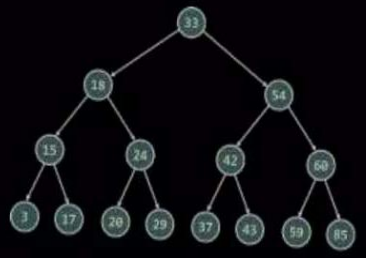
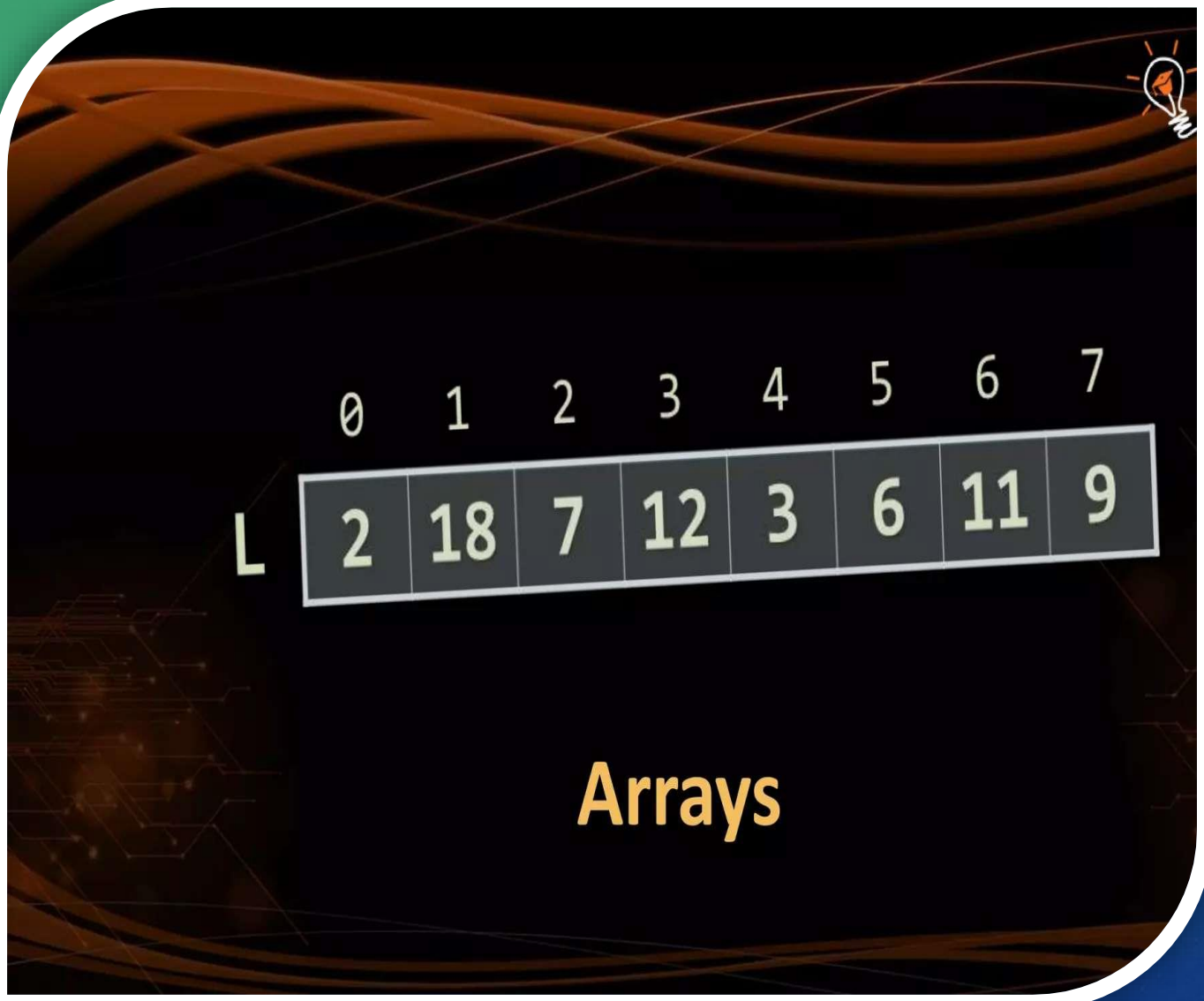


Table of Contents

1. Arrays
   - `int[]`, `String[]`, etc.
2. Lists
   - `ArrayList<E>`
3. Strings
   - `String str = "Hello";`
4. Sets
   - `HashSet<E>`, `TreeSet<E>`
5. Maps
   - `HashMap<K, V>`, `TreeMap<K, V>`

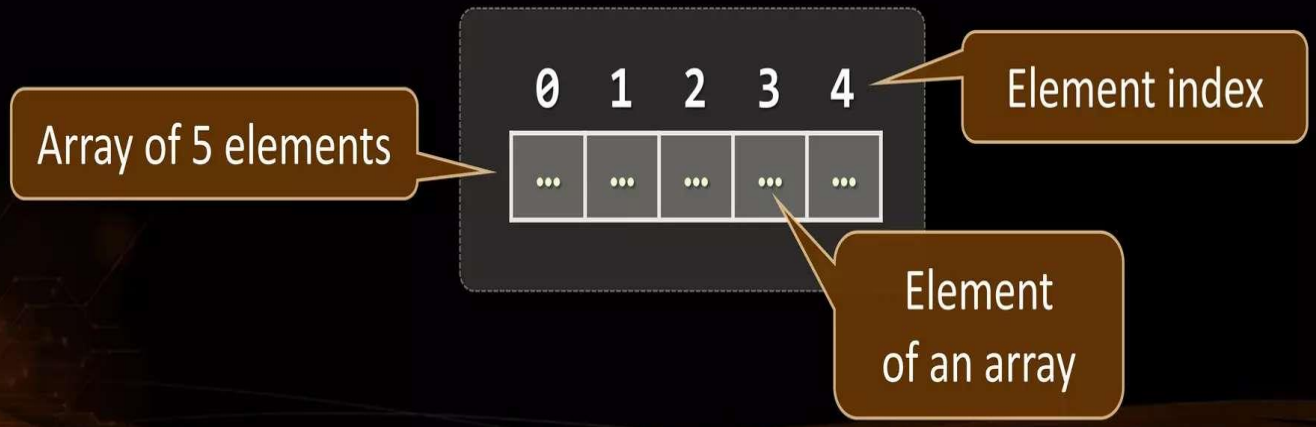
Warning: Not for Absolute Beginners

- The "**Java Basics**" course is NOT for absolute beginners
  - Take the "C# Basics" course at SoftUni first: https://softuni.bg/courses/csharp-basics
  - The course is for beginners, but with previous coding skills
- Requirements
  - Coding skills – entry level
  - Computer English – entry level
  - Logical thinking


0 1 2 3 4 5 6 7
L
2 18 7 12 3 6 11 9
Arrays

What are Arrays?

- In programming array is a sequence of elements
  - All elements are of the same type
  - The order of the elements is fixed
  - Has fixed size (`length`)



Working with Arrays in Java

- Allocating an array of 10 integers:

```
int[] numbers = new int[10];
```

- Assigning values to the array elements:

```
for (int i=0; i<numbers.length; i++)
    numbers[i] = i+1;
```

- Accessing array elements by index:

```
numbers[3] = 20;
numbers[5] = numbers[2] + numbers[7];
```

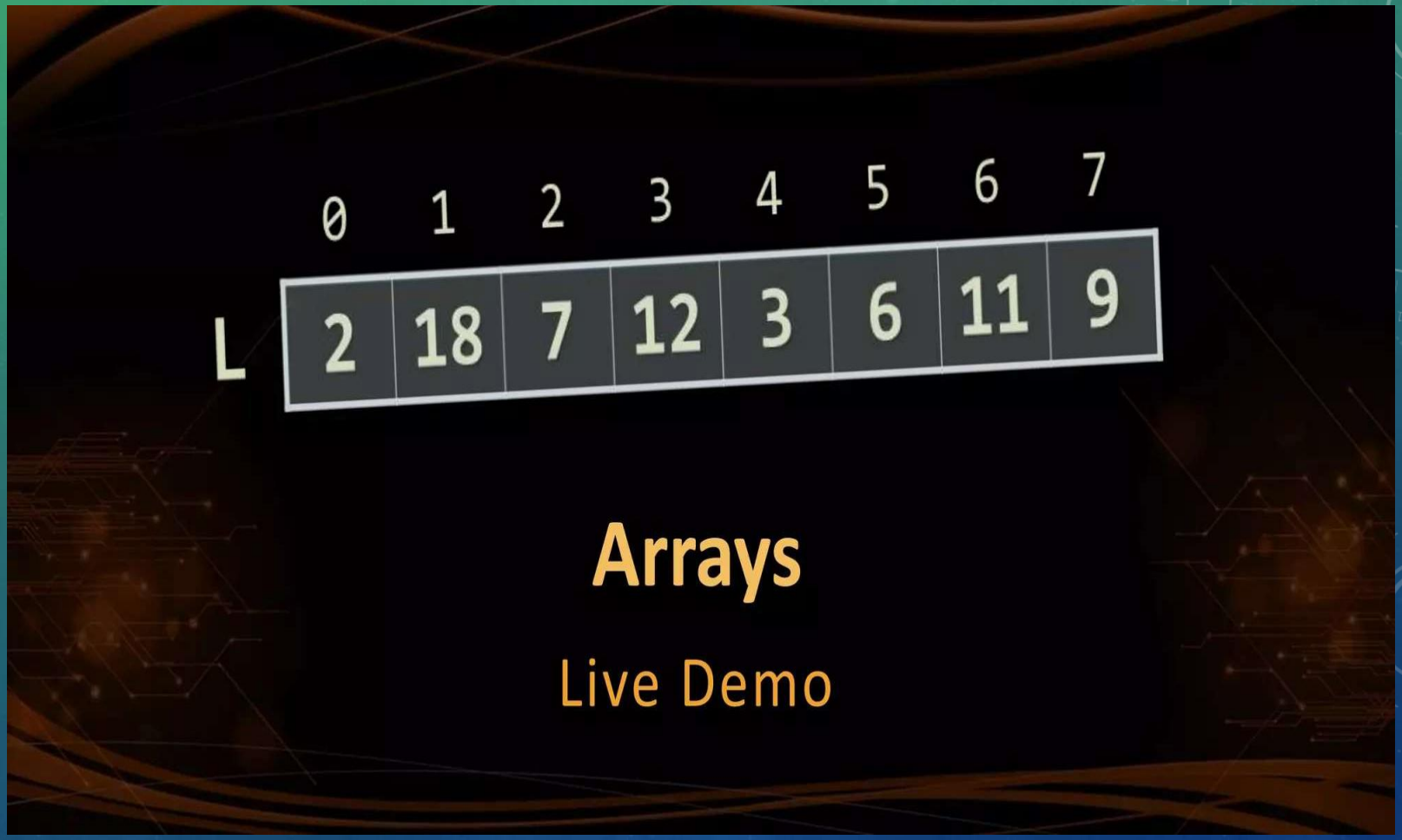
Arrays of Strings

- You may define an array of any type, e.g. **String**:

```
String[] names = { "Peter", "Maria", "Katya", "Todor" };
for (int i = 0; i<names.length; i++) {
  System.out.printf("names[%d] = %s\n", i, names[i]);
}
for (String name : names) {
  System.out.println(name);
}

names[4] = "Nakov"; // ArrayIndexOutOfBoundsException
names.length = 5; // array.length is read-only field
```

Read, Sort and Print Array of n Strings

```
Scanner scanner = new Scanner(System.in);
int n = scanner.nextInt();
scanner.nextLine();
String[] lines = new String[n];
for (int i = 0; i < n; i++) {
  lines[i] = scanner.nextLine();
}

Arrays.sort(lines);

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


0 1 2 3 4 5 6 7
L
2 18 7 12 3 6 11 9
Arrays
Live Demo


7
1 2 3 4 5 6
Lists
Using ArrayList<E>

Lists in Java

- In Java arrays have fixed length
  - Cannot add / remove / insert elements
- Lists are like resizable arrays
  - Allow add / remove / insert of elements
- Lists in Java are defined through the **ArrayList<E>** class
  - Where **E** is the type of the list, e.g. **String** or **Integer**

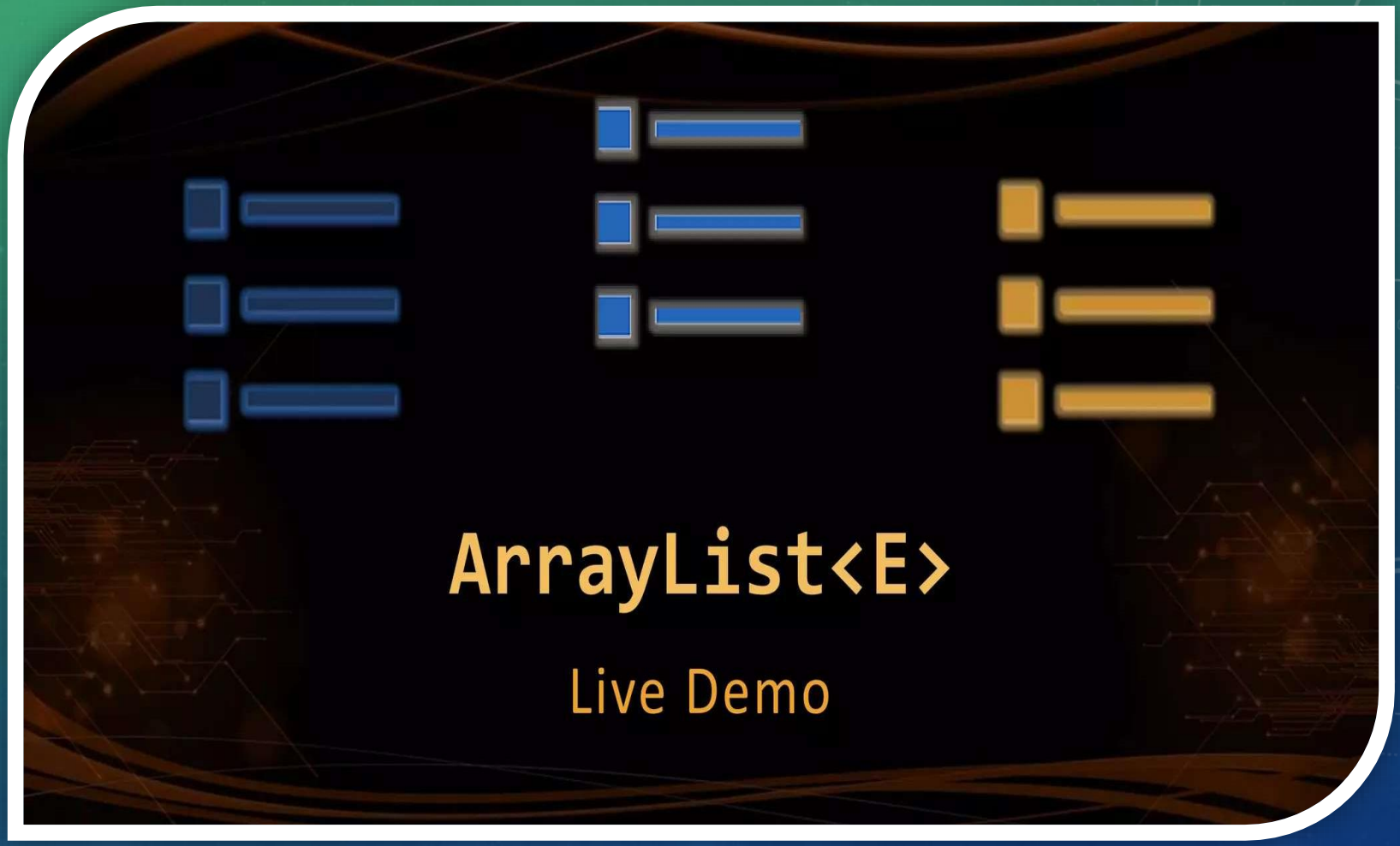
```
ArrayList<Integer> numbers = new ArrayList<Integer>();
numbers.add(5);
System.out.println(numbers.get(0)); // 5
```

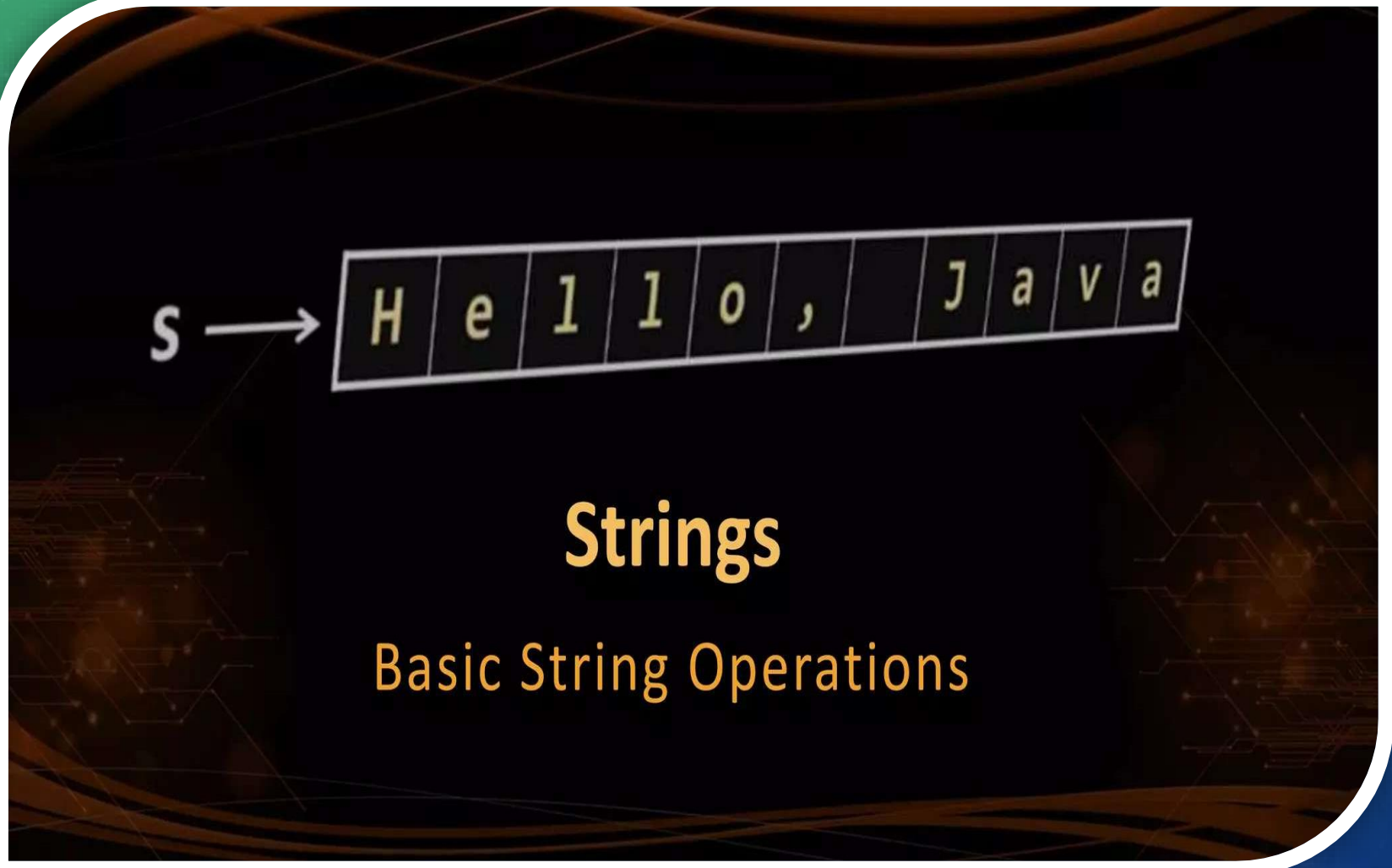
ArrayList<String> – Example

```
ArrayList<String> names = new ArrayList<String>() {{
        add("Peter");
        add("Maria");
        add("Katya");
        add("Todor");
}};
names.add("Nakov"); // Peter, Maria, Katya, Todor, Nakov
names.remove(0); // Maria, Katya, Todor, Nakov
names.remove(1); // Maria, Todor, Nakov
names.remove("Todor"); // Maria, Nakov
names.addAll(Arrays.asList("Alice", "Tedy"));
   // Maria, Nakov, Alice, Tedy
names.add(3, "Sylvia"); // Maria, Nakov, Alice, Sylvia, Tedy
names.set(2, "Mike"); // Maria, Nakov, Mike, Sylvia, Tedy
System.out.println(names);
```

ArrayList<Integer> – Example

```
// This will not compile!
ArrayList<int> intArr = new ArrayList<int>();

ArrayList<Integer> nums = new ArrayList<>(
    Arrays.asList(5, -3, 10, 25));
nums.add(55); // 5, -3, 10, 25, 55
System.out.println(nums.get(0)); // 5
System.out.println(nums); // [5, -3, 10, 25, 55]
nums.remove(2); // 5, -3, 25, 55
nums.set(0, 101); // 101, -3, 25, 55
System.out.println(nums); // [101, -3, 25, 55]
```


ArrayList<E>
Live Demo


s
H e l l o , J a v a
Strings
Basic String Operations

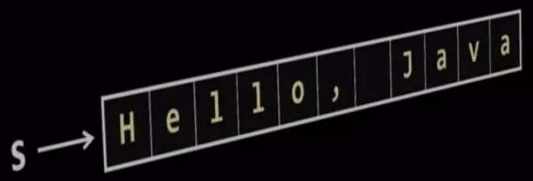
What Is String?

- Strings are indexed sequences of Unicode characters
  - Represented by the **`String`** class in Java
  - Characters accessed by index: **`0`** … **`length()-1`**
- Example:

```
string s = "Hello, SoftUni!";
```

Working with Strings

- Strings in Java
  - Know their number of characters: `length()`
  - Can be accessed by index: `charAt(0` … `length()-1)`
  - Reference types
    - Stored in the heap (dynamic memory)
    - Can have `null` value (missing value)



- Strings cannot be modified (immutable)
  - Most string operations return a new `String` instance
  - `StringBuilder` class is used to build stings

Strings – Examples

```
String str = "SoftUni";

System.out.println(str);
for (int i = 0; i < str.length(); i++) {
    System.out.printf("str[%d] = %s\n", i, str.charAt(i));
}

System.out.println(str.indexOf("Uni")); // 4
System.out.println(str.indexOf("uni")); // -1 (not found)
System.out.println(str.substring(4, 7)); // Uni
System.out.println(str.replace("Soft", "Hard")); // HardUni
System.out.println(str.toLowerCase()); // softuni
System.out.println(str.toUpperCase()); // SOFTUNI
```

Strings – Examples (2)

```
String firstName = "Steve";
String lastName = "Jobs";
int age = 56;
System.out.println(firstName + " " + lastName +
    " (age: " + age + ")"); // Steve Jobs (age: 56)
String allLangs = "C#, Java; HTML, CSS; PHP, SQL";
String[] langs = allLangs.split("[, ;]+");
for (String lang : langs) {
    System.out.println(lang);
}
System.out.println("Langs = " + String.join(", ", langs));
System.out.println("  \n\n Software   University  ".trim());
```

Comparing Strings in Java

- The == operator does not work correctly for strings!
  - Use `String.equals(String)` and `String.compareTo(String)`

```
String[] words = "yes yes".split(" ");
System.out.println("words[0] = " + words[0]); // yes
System.out.println("words[1] = " + words[0]); // yes
System.out.println(words[0] == words[1]); // false
System.out.println(words[0].equals(words[1])); // true
System.out.println("Alice".compareTo("Mike")); // < 0
System.out.println("Alice".compareTo("Alice")); // == 0
System.out.println("Mike".compareTo("Alice")); // > 0
```

Regular Expressions

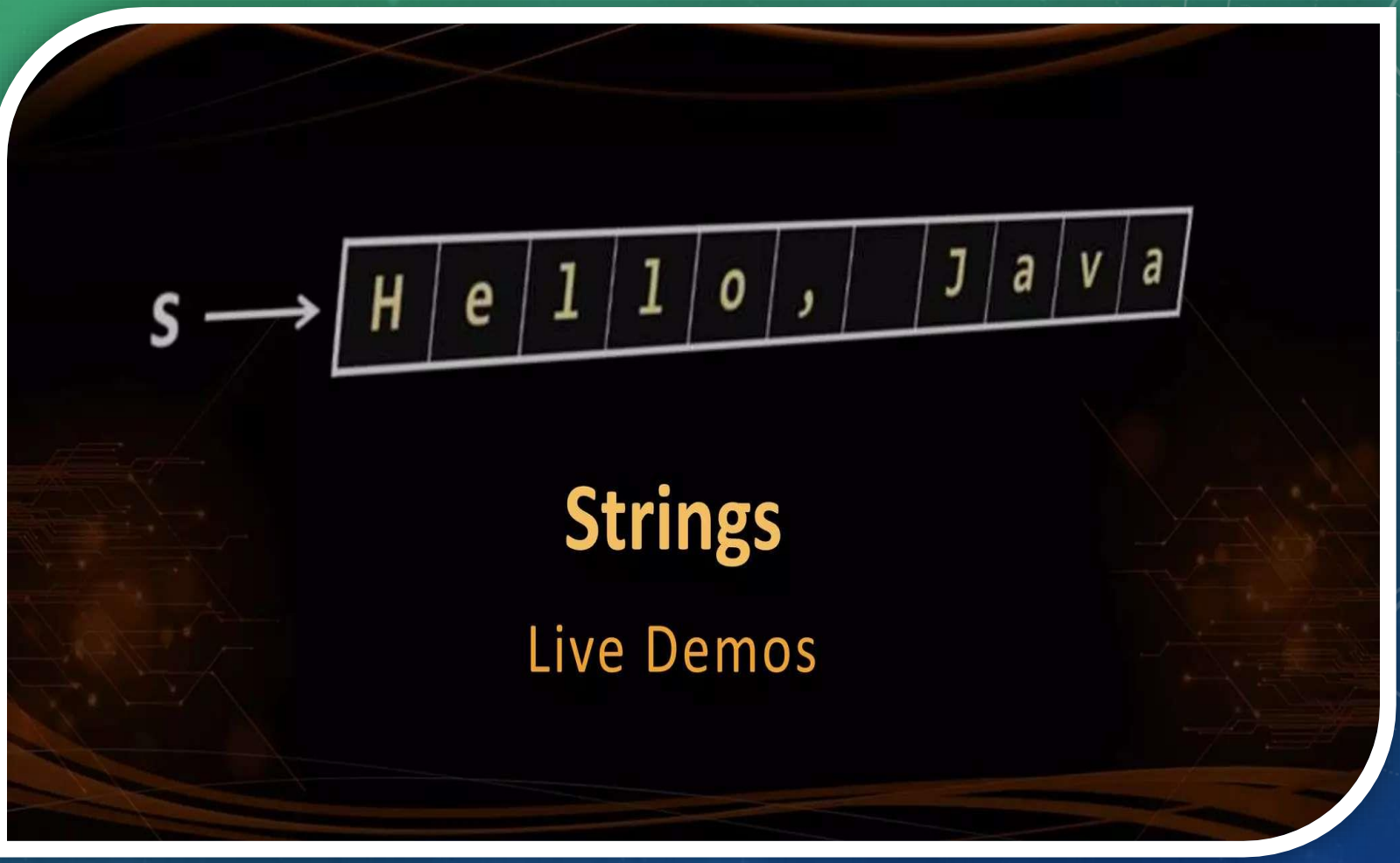
- Regular expressions match text by pattern, e.g.
  - `[0-9]+` matches a non-empty sequence of digits
  - `[a-zA-Z]*` matches a sequence of letters (including empty)
  - `[A-Z][a-z]+ [A-Z][a-z]+` matches a name (first name + space + last name)
  - `\s+` matches any whitespace; `\S+` matches non-whitespace
  - `\d+` matches digits; `\D+` matches non-digits
  - `\w+` matches letters (Unicode); `\W+` matches non-letters
  - `\+\d{1,3}([ -]*[0-9]+)+` matches international phone

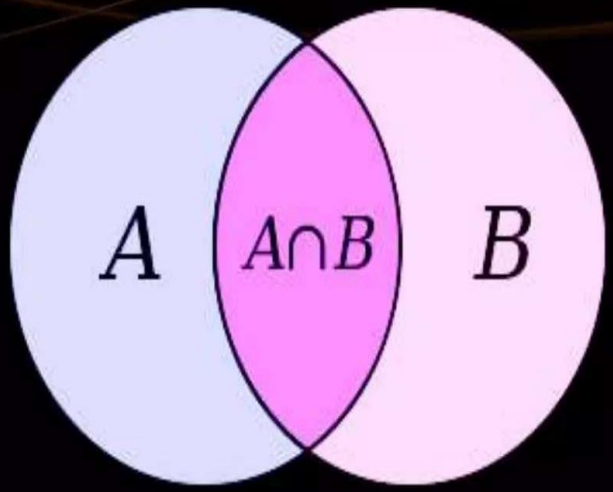
Validation by Regular Expression – Example

```
import java.util.regex.*;
…

String regex = "\\+\\d{1,3}([ -]*[0-9]+)+";
System.out.println("+359 2 981-981".matches(regex)); // true
System.out.println("invalid number".matches(regex)); // false
System.out.println("+359 123-".matches(regex)); // false
System.out.println("+359 (2) 981 981".matches(regex)); // false
System.out.println("+44 280 11 11".matches(regex)); // true
System.out.println("++44 280 11 11".matches(regex)); // false
System.out.println("(+49) 325 908 44".matches(regex)); // false
System.out.println("+49 325 908-40-40".matches(regex)); // true
```

Find Matches by Pattern – Example

```
import java.util.regex.*;
…
String text =
    "Hello, my number in Sofia is +359 894 11 22 33, " +
    "but in Munich my number is +49 89 975-99222.";
Pattern phonePattern = Pattern.compile(
    "\\+\\d{1,3}([ -]*([0-9]+))+");
Matcher matcher = phonePattern.matcher(text);
while (matcher.find()) {
    System.out.println(matcher.group());
}
// +359 894 11 22 33
// +49 89 975-99222
```


s
H e l l o , J a v a
Strings
Live Demos



Sets

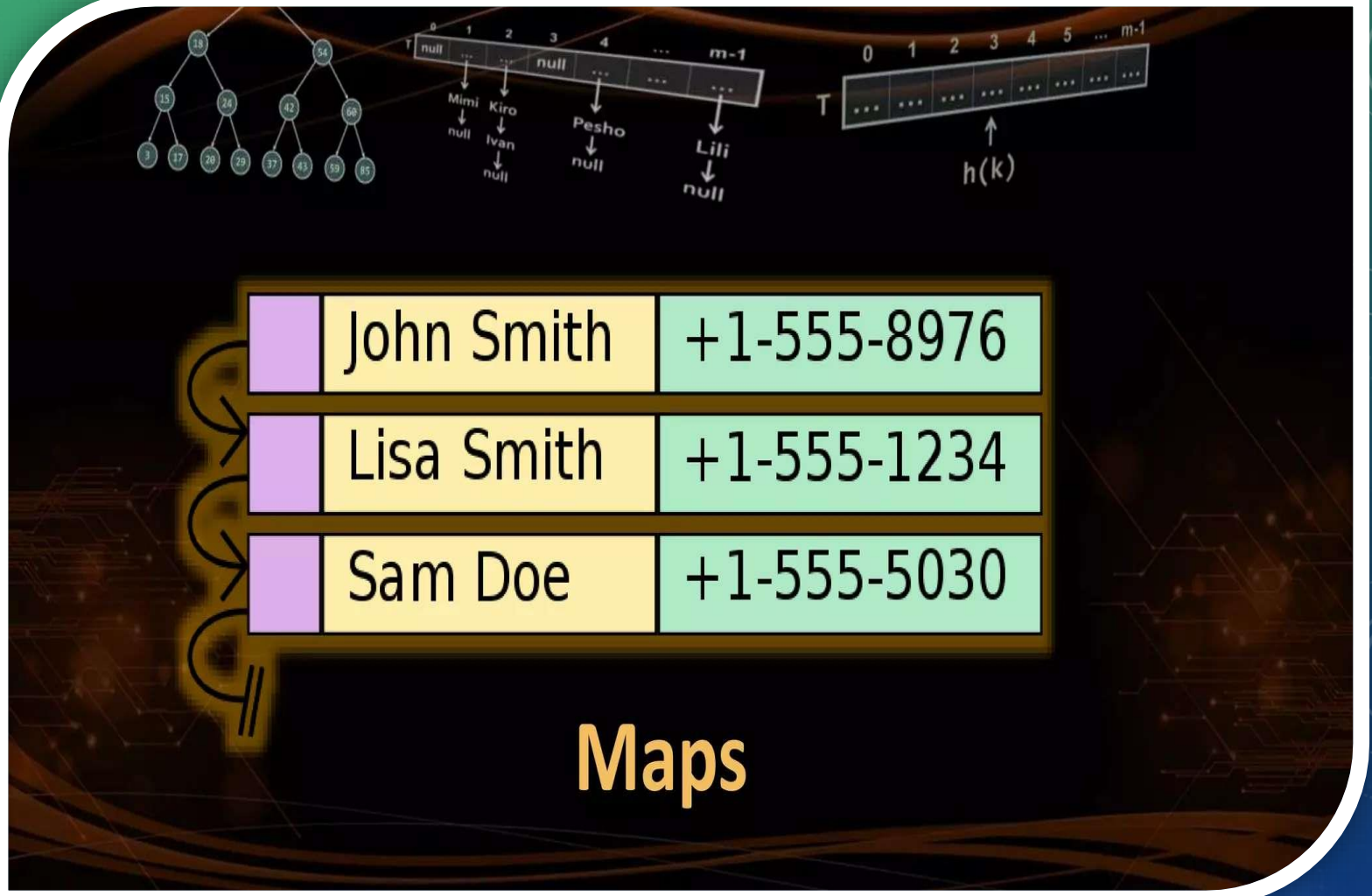
HashSet<E> and TreeSet<E>

Sets in Java

- Sets in Java keep unique elements
  - Like lists but duplicated elements are stored only once
- `HashSet<E>`
  - Keeps a set of elements in a hash-tables
  - The elements are randomly ordered (by their hash code)
- `TreeSet<E>`
  - Keeps a set of elements in a red-black ordered search tree
  - The elements are ordered incrementally

HashSet<E> and TreeSet<E> – Examples

```
Set<String> set = new TreeSet<String>();
set.add("Pesho");
set.add("Tosho");
set.add("Pesho");
set.add("Gosho");
set.add("Maria");
set.add("Alice");
set.remove("Pesho");
System.out.println(set); // [Alice, Gosho, Maria, Tosho]
```


John Smith
+1-555-8976
Lisa Smith
+1-555-1234
Sam Doe
+1-555-5030
Maps

Maps in Java

- Maps in Java keep unique <key, value> pairs
- `HashMap<K, V>`
  - Keeps a map of elements in a hash-table
  - The elements are randomly ordered (by their hash code)
- `TreeMap<K, V>`
  - Keeps a set of elements in a red-black ordered search tree
  - The elements are ordered incrementally by their key

HashMap<K, V> – Examples

- Counting words occurrences in a list:

```
String[] words = { "yes", "hi", "hello", "hi", "welcome",
  "yes", "yes", "welcome", "hi", "yes", "hello", "yes" };

Map<String, Integer> wordsCount = new HashMap<String, Integer>();
for (String word : words) {
  Integer count = wordsCount.get(word);
  if (count == null) {
    count = 0;
  }
  wordsCount.put(word, count+1);
}

System.out.println(wordsCount); // {hi=3, yes=5, hello=2, welcome=2}
```

TreeMap<K, V> – Examples

- Students and their grades

```
HashMap<String, ArrayList<Integer>> grades = new HashMap<>();
grades.put("Peter", new ArrayList<>(Arrays.asList(5)));
grades.put("George", new ArrayList<>(Arrays.asList(5, 5, 6)));
grades.put("Maria", new ArrayList<>(Arrays.asList(5, 4, 4)));
grades.get("Peter").add(6);
grades.get("George").add(6);

for (String key : grades.keySet()) {
  System.out.println("" + key + " -> " + grades.get(key));
}
```

Summary

- Arrays, Strings and Collections:
  1. Arrays: `int[]`, `String[]`, etc.
  2. Strings: `String str = "Hello";`
  3. Lists: `ArrayList<E>`
  4. Sets: `HashSet<E>`, `TreeSet<E>`
  5. Maps: `HashMap<K, V>`, `TreeMap<K, V>`

WEEK 13-15

Introduction

- Object Oriented Programming (OOP)
- Structured Programming

- Programming paradigm is a fundamental style of computer programming

Introduction

- Programming paradigms differ in how each element of the programs is represented and how steps are defined for solving problems.
- OOP focuses on representing problems using real-world objects and their behaviour, while Structured Programming deals with organizing the program in a logical structure.

What is Structured/ Procedural Programming?

- A structured program is made up of simple program flow structures, which are hierarchically organized.
- They are Sequence, Selection And Repetition.

What is Object Oriented Programming?

- thinking about the problem to be solved in terms of real-world elements and representing the problem in terms of objects and their behaviour
- Classes depict the abstract representations of real world objects

What is Object Oriented Programming?

- Classes have properties called attributes.
- Attributes are implemented as global and instance variables.
- Methods in the classes represent or define the behaviour of these classes.
- Methods and attributes of classes are called the members of the class.
- An instance of a class is called an object.

What is Object Oriented Programming?

There are several important OOP concepts such as

- Data abstraction,
- Encapsulation,
- Polymorphism,
- Messaging,
- Modularity and
- Inheritance.

What is Object Oriented Programming?

Structured/Procedural Programming	Object Oriented Programming
Structured Programming is designed which focuses on process/ logical structure and then data required for that process.	Object Oriented Programming is designed which focuses on data.
Structured programming follows top-down approach.	Object oriented programming follows bottom-up approach.
Structured Programming is also known as Modular Programming and a subset of procedural programming language.	Object Oriented Programming supports inheritance, encapsulation, abstraction, polymorphism, etc.
In Structured Programming, Programs are divided into small self-contained functions.	In Object Oriented Programming, Programs are divided into small entities called objects.
Structured Programming is less secure as there is no way of data hiding.	Object Oriented Programming is more secure as having data hiding feature.
Structured Programming can solve moderately complex programs.	Object Oriented Programming can solve any complex programs.
Structured Programming provides less reusability, more function dependency.	Object Oriented Programming provides more reusability, less function dependency.
Less abstraction and less flexibility.	More abstraction and more flexibility.

OOPs Basic concepts

- OOP allow decomposition of a problem into a number of entities called **Object** and then builds **data and function** around these objects.
  - The data of the objects can be accessed only by the functions associated with that object.
  - The functions of one object can access the functions of other object.

OOPs Basic concepts...Class:

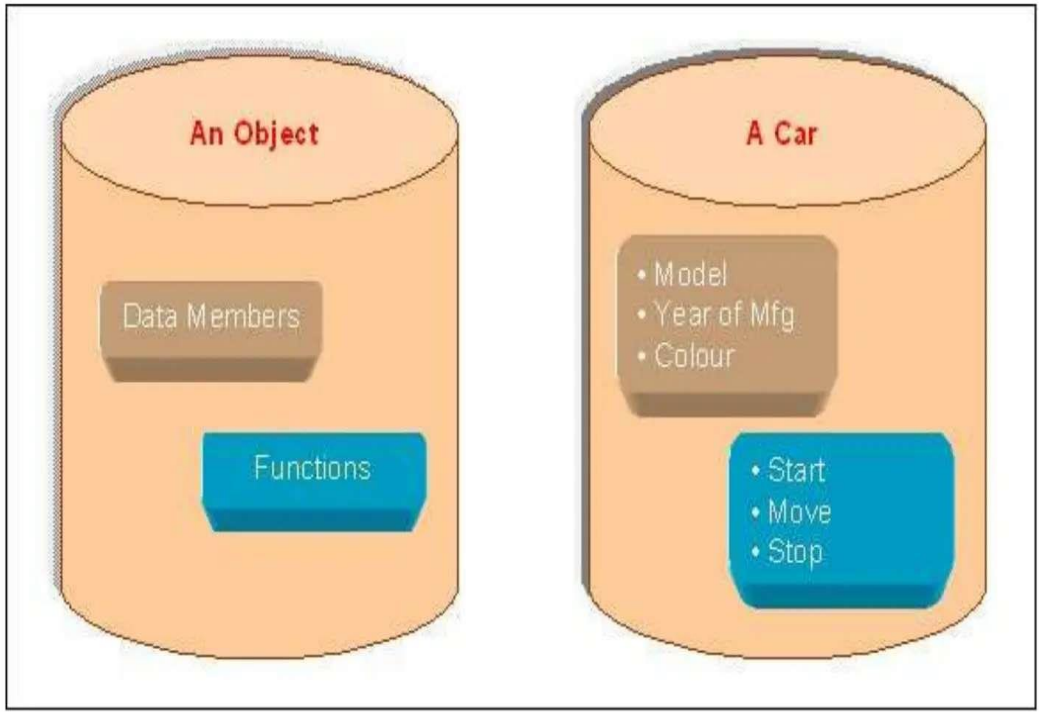
- Class is a blueprint of an object that contains variables for storing data and functions to performing operations on these data.
- Class will not occupy any memory space and hence it is only logical representation of data.

```
class Employee
{

}
```

OOPs Basic concepts...Object:

- Objects are the basic run-time entities in an object oriented system. They may represent a person, a place or any item that the program has to handle.
- "Object is a Software bundle of related variable and methods."
- “Object is an instance of a class”

OOPs Basic concepts...Object:



OOPs Basic concepts...Object:

- **Class will not occupy any memory space.** Hence to work with the data represented by the class **you must create a variable for the class**, which is called as an **object.**

OOPs Basic concepts...Object:

- When an object is created by using the **keyword new**, then **memory will be allocated** for the class **in heap memory area**, which is called as an instance and its starting address will be stored in the **object in stack memory area**.
- When an object is created **without the keyword new,** then **memory will not be allocated** in heap i.e. instance will not be created and object in the stack **contains the value null.**

OOPs Basic concepts...Object:

- When an object contains null, then it is not possible to access the members of the class using that object.

```
class Employee
{

}
```

- Syntax to create an object of class Employee:-

```
Employee objEmp = new Employee();
```

OOPs Basic concepts...

- Abstraction
- Encapsulation
- Inheritance
- Polymorphism

OOPs Basic concepts...Abstraction

- Abstraction is "To represent the essential feature without representing the back ground details."
- Abstraction lets you focus on what the object does instead of how it does it.
- Abstraction provides you a generalized view of your classes or object by providing relevant information.
- Abstraction is the process of hiding the working style of an object, and showing the information of an object in understandable manner.

OOPs Basic concepts...Abstraction

- Real world Example of Abstraction: -
- Suppose you have an object Mobile Phone.
- Suppose you have 3 mobile phones as following:-

Nokia 1400 (Features:- Calling, SMS)

Nokia 2700 (Features:- Calling, SMS, FM Radio, MP3, Camera)

Black Berry (Features:-Calling, SMS, FM Radio, MP3, Camera, Video Recording, Reading E-mails)

OOPs Basic concepts...Abstraction

Abstract information (Necessary and Common Information) for the object "Mobile Phone" is make a call to any number and can send SMS." So that, for mobile phone object you will have abstract class like following:-

```
abstract class MobilePhone
 {
    public void Calling();
    public void SendSMS();
 }

 public class Nokia1400 : MobilePhone
 {

 }

 public class Nokia2700 : MobilePhone
 {
    public void FMRadio();
    public void MP3();
    public void Camera();
 }
```

```
public class BlackBerry : MobilePhone
  {
     public void FMRadio();
     public void MP3();
     public void Camera();
     public void Recording();
     public void ReadAndSendEmails();

  }
```

OOPs Basic concepts...Encapsulation

- Wrapping up data member and method together into a single unit (i.e. Class) is called Encapsulation.
- That is enclosing the related operations and data related to an object into that object.

```
class Teacher
  {
    int id;
    string name;
    void teaching();
  }
```

OOPs Basic concepts...Encapsulation

```
class Demo
{
      private int mark;
      public int Mark
     {
            get { return mark; }
            set {   if (mark > 0)
                        mark = value; else _mark = 0;
                  }
     }
 }
```

OOPs Basic concepts...Inheritance

- When a class acquire the property of another class is known as inheritance.
- Inheritance is process of object reusability.

OOPs Basic concepts...Inheritance

```
public class ParentClass
{
        public ParentClass()
        {
            Console.WriteLine("Parent Constructor.");
        }

        public void print()
        {
            Console.WriteLine("I'm a Parent Class.");
        }
}
```

```
public class ChildClass : ParentClass
{
      public ChildClass()
      {
        Console.WriteLine("Child Constructor.");
      }

      public static void Main()
      {
        ChildClass child = new ChildClass();
        child.print();
      }
}
```

OOPs Basic concepts...Polymorphism

- Polymorphism means one name many forms.
- One function behaves different forms. .
- In other words, "Many forms of a single object is called Polymorphism."

Difference between Abstraction and Encapsulation

Abstraction	Encapsulation
1. Abstraction solves the problem in the design level.	1. Encapsulation solves the problem in the implementation level.
2. Abstraction is used for hiding the unwanted data and giving relevant data.	2. Encapsulation means hiding the code and data into a single unit to protect the data from outside world.
3. Abstraction lets you focus on what the object does instead of how it does it	3. Encapsulation means hiding the internal details or mechanics of how an object does something.
4. Abstraction- Outer layout, used in terms of design. For Example:- Outer Look of a Mobile Phone, like it has a display screen and keypad buttons to dial a number.	4. Encapsulation- Inner layout, used in terms of implementation. For Example:- Inner Implementation detail of a Mobile Phone, how keypad button and Display Screen are connect with each other using circuits.

WEEK 16 & 17

PROJECT PLANNING, PRESENTATION, PROJECT SHOW

THANK YOU