Core Java

Agenda

- Introduction
- Data Types
- Operators
- Flow Control
- Arrays and Strings

Programming Languages

- Machine Language
- Assembly Language
- High level Language

Machine Language

- The fundamental language of the computer’s processor, also called Low Level Language.
- All programs are converted into machine language before they can be executed.
- Consists of combination of 0’s and 1’s that represent high and low electrical voltage.

Disadvantages:

1. Very difficult to write, because:

    Binary system – not user friendly to human

    It requires excellent memorizing power

2. Programmer has to keep track of storage locations of data and instruction

Assembly Language:

Uses **symbolic** operation code to represent the machine operation code.

Instruction codes are represented by **mnemonics** (a set of letters)

Disadvantage:

Similar to that of Machine level language

- Machine dependent
- The program is usually long
- Hard to learn and slow to write

High Level Language

- Computer (programming) languages that are easier to learn.
- Uses English like statements.
- Examples are C ++, Visual Basic, Pascal, Fortran and Java

Advantages:

1. Easier to write, to read & to modify

   -written in English-like format

2. Programs – faster & shorter to code

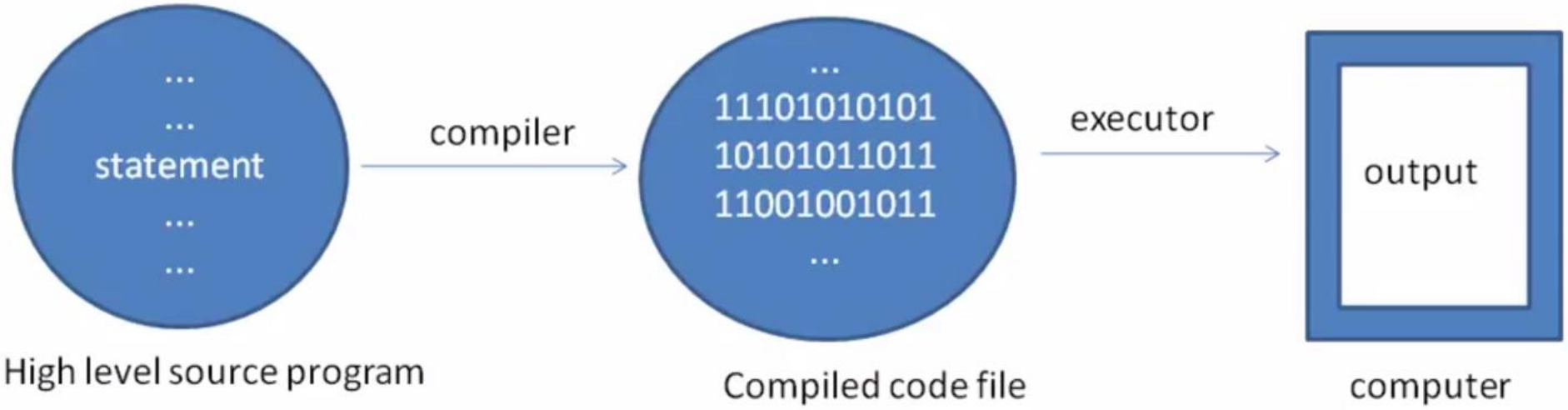
   - One statement for several computer operations

3. More portable, i.e. can be executed by different computers

   - machine independent

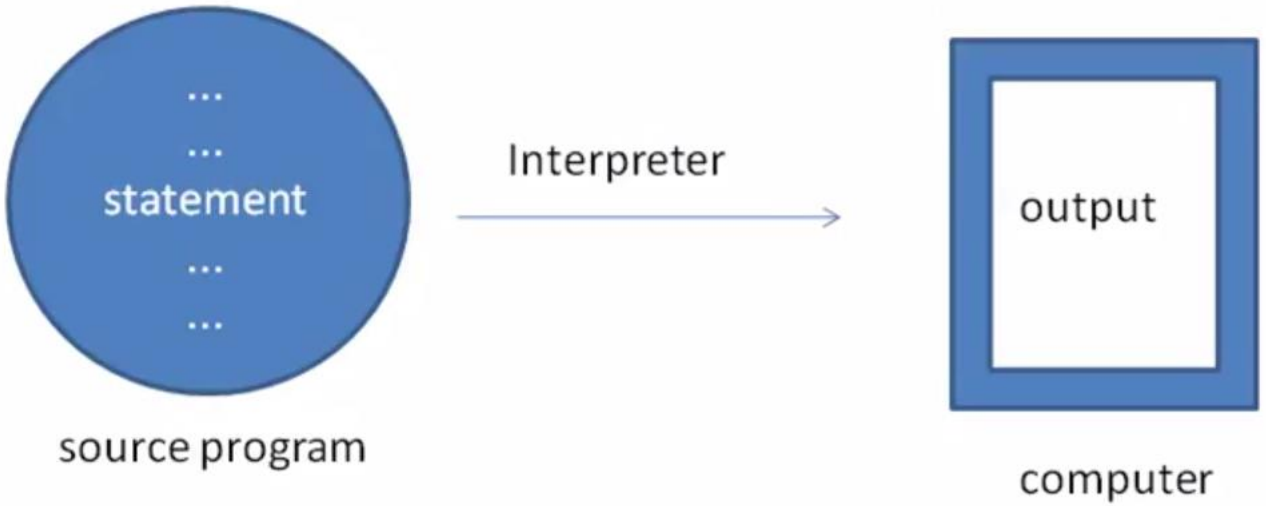
What is a compiler

A compiler translates the entire source code into a machine-code file or any intermediate code, and that file is then executed.



What a Interpreter

An interpreter reads one statement from the source code, translates it to the machine code or virtual machine code, and then executes it right away.



Object Oriented Language:

- Object-oriented language supports all the features of OOPs.
  - Object
  - Class
  - Encapsulation
  - Abstraction
  - Inheritance
  - Polymorphism
- Object-oriented language doesn't has in-built object.
- Object-oriented languages are C++, C#, Java etc.

Object Based Language:

- Object-based language doesn't support all the features of OOPs like Polymorphism and Inheritance
- Object-based language has in-built object like javascript has window object.
- Object-based languages are Javascript, VB etc.

Introduction – What is Java

- Programming language
  - Another programming language using which we can develop applets, standalone applications, web applications and enterprise applications.
- Platform Independent
  - A Java program written and compiled on one machine can be executed on any other machine (irrespective of the operating system)
- Object Oriented
  - Complies to object oriented programming concepts. Your program is not object oriented unless you code that way
- Compiled and Interpreted
  - The .java file is compiled to a .class file & the .class file is interpreted to machine code

- Secure

  With Java's secure feature, it enables to develop virus-free, tamper-free systems. Authentication techniques are based on public-key encryption.

- Portable

- Robust

  Java makes an effort to eliminate error prone situations by emphasizing mainly on compile time error checking and runtime checking.

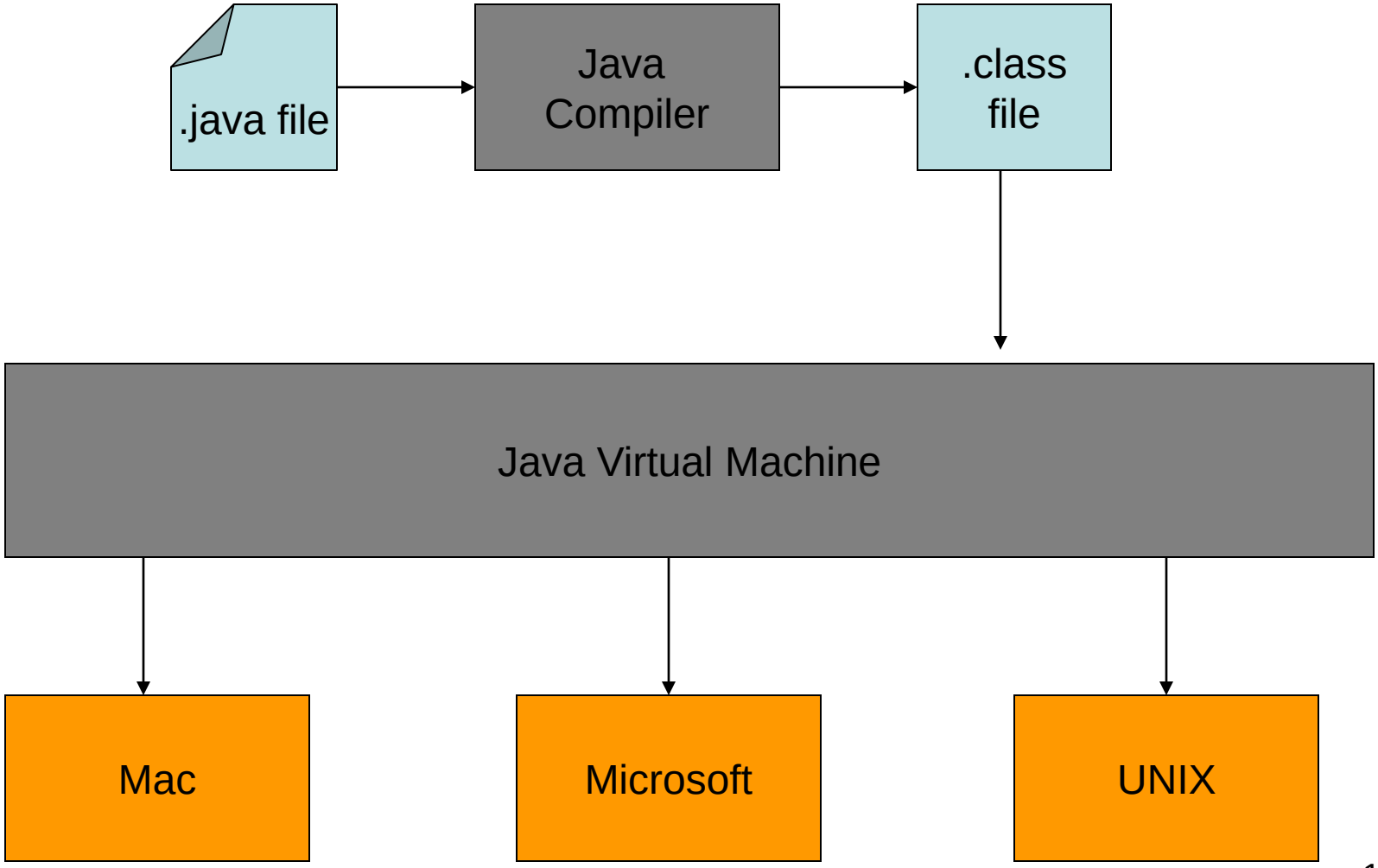
- Multithreaded

  With Java's multithreaded feature, it is possible to write programs that can do many tasks simultaneously.

- Distributed

  Java is designed for the distributed environment of the internet.

Introduction – Java Virtual Machine



The “**JDK**” is the Java Development Kit. I.e., the JDK is bundle of software that you can use to develop Java based software.

The "**JRE**" is the Java Runtime Environment. I.e., the JRE is an implementation of the Java Virtual Machine which actually executes Java programs.

Javac : The Java compiler. Converts Java source code into byte codes.

Java : The Java interpreter. Executes Java application byte codes directly from class files.

Introduction – My First Program Version 1

```
package com.altisource.test;

public class HelloWorld
{
  public static void main(String[] args)
  {
    System.out.println(“Hello World”);
  }
}
```

Compile the program: javac HelloWorld.java
Execute the program: java HelloWorld
Output: Hello World

Introduction – My First Program Version 2

```
package com.altisource.test;

public class HelloWorld
{
  public static void main(String[] args)
  {
    HelloWorld hw = new HelloWorld();
    hw.display();
  }

  public void display()
  {
    System.out.println(“Hello World”);
  }
}
```

Compile the program: javac HelloWorld.java
Execute the program: java HelloWorld
Output: Hello World

Introduction – My First Program Version 3

```
package com.altisource.test;

public class HelloWorld
{
  public static void main(String[] args)
  {
    HelloWorld hw = new HelloWorld();
    hw.display(args[0]);
  }

  public void display(String userName)
  {
    System.out.println("Hello " + userName);
  }
}
```

Compile the program: javac HelloWorld.java
Execute the program: java HelloWorld Apparao
Output: Hello Apparao

Introduction – My First Program Version 4

```
package com.altisource.test;

public class HelloWorld
{
  String userName;
  public static void main(String[] args)
  {
    HelloWorld hw = new HelloWorld();
    hw.userName = args[0];
     hw.display();
  }

  public void display()
  {
    System.out.println("Hello " + userName);
  }
}
```

Compile the program: javac HelloWorld.java
Execute the program: java HelloWorld Sharad
Output: Hello Sharad

Introduction – Java Keywords

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

Variable

- Variables are nothing but reserved memory locations to store values.
- This means that when you create a variable you reserve some space in memory.
- Based on the data type of a variable, the operating system allocates memory and decides what can be stored in the reserved memory.

Example: **dataType variableName ;**

Following are the rules for variable declaration:

1. The variable name should not start with digits.
2. No special character is allowed in identifier except underscore.
3. There should not be any blank space with the identifier name.
4. The identifier name should not be a keyword.
5. The identifier name should be meaningful.

Introduction - Data Types

Data type	Bytes	Min Value	Max Value	Default Value
byte	1	-2^7	$2^7 - 1$	0
short	2	-2^{15}	$2^{15} - 1$	0
int	4	-2^{31}	$2^{31} - 1$	0
long	8	-2^{63}	$2^{63} - 1$	0L
float	4	-	-	0.0f
double	8	-	-	0.0d
char	2	0	$2^{16} - 1$	
boolean	-	-	-	false

General rule:
Min value = $2^{(bits - 1)}$
Max value = $2^{(bits-1)} - 1$
(where 1 byte = 8 bits)

Operators - Types

- Definition:

  An operator performs a particular operation on the operands it is applied on

- Types of operators
  - Assignment Operators
  - Arithmetic Operators
  - Unary Operators
  - Equality Operators
  - Relational Operators
  - Conditional Operators
  - instaceof Operator
  - Bitwise Operators
  - Shift Operators

Operators – Assignment Operators/Arithmetic Operators

- Assignment Operator

Operator	Description	Example
=	Assignment	int i = 10; int j = i;

- Arithmetic Operators

Operator	Description	Example
+	Addition	int i = 8 + 9; byte b = (byte) 5+4;
-	Subtraction	int i = 9 – 4;
*	Multiplication	int i = 8 * 6;
/	Division	int i = 10 / 2;
%	Remainder	int i = 10 % 3;

Operators – Unary Operators/Equality Operators

- Unary Operators

Operator	Description	Example
+	Unary plus	int i = +1;
-	Unary minus	int i = -1;
++	Increment	int j = i++;
--	Decrement	int j = i--;
!	Logical Not	boolean j = !true;

- Equality Operators

Operator	Description	Example
==	Equality	If (i==1)
!=	Non equality	If (i != 4)

Operators – Relational Operators/Conditional Operators

- Relational Operators

Operator	Description	Example
>	Greater than	if (x > 4)
<	Less than	if (x < 4)
>=	Greater than or equal to	if (x >= 4)
<=	Less than or equal to	if (x <= 4)

- Conditional Operators

Operator	Description	Example
&&	Conditional and	If (a == 4 && b == 5)
||	Conditional or	If (a == 4 || b == 5)

Operators – instanceof Operator/Bitwise Operators/shift operators

- instanceof Operator

Operator	Description	Example
instanceof	Instamce of	If (john instance of person)

- Bitwise Operators

Operator	Description	Example
&	Bitwise and	001 & 111 = 1
|	Bitwise or	001 | 110 = 111
^	Bitwise ex-or	001 ^ 110 = 111
~	Reverse	~011 = -10

- Shift Operators

Operator	Description	Example
>>	Right shift	4 >> 1 = 100 >> 1 = 010 = 2
<<	Left Shift	4 << 1 = 100 << 1 = 1000 = 8
>>>	Unsigned Right shift	4 >>> 1 = 100 >>> 1 = 010 = 2

Ternary operator

```
result = value > conditionValue ? result1 : result2
public class Test {
    public static void main(String[] args) {
      int age = 18;
      System.out.println(
        i > 18 ? ”You are eligible for voter card"
                : “You are not eligible for voter card");

    }
  }
```

Comma Operator

```
public class Test {
   public static void main(String[] args) {
      for(int i = 1, j = i + 10; i < 5; i++, j = i * 2) {
        System.out.println("i= " + i + " j= " + j);
      }
    }
   }
```

Flow Control – if-else

- **if-else**

Syntax	Example

if (<condition-1>) {
// logic for true condition-1 goes here
} else if (<condition-2>) {
// logic for true condition-2 goes here
} else {
// if no condition is met, control comes here
}
	int a = 10;
	if (a < 10) {
	System.out.println(“Less than 10”);
	} else if (a > 10) {
	System.out.pritln(“Greater than 10”);
	} else {
	System.out.println(“Equal to 10”);
	}
	
	Result: Equal to 10s

Flow Control – switch

- **switch**

Syntax	Example

switch (<value>) {
case <a>:
// stmt-1
break;
case <b>:
//stmt-2
break;
default:
//stmt-3
	int a = 10;
	switch (a) {
	case 1:
	System.out.println(“1”);
	break;
	case 10:
	System.out.println(“10”);
	break;
	default:
	System.out.println(“None”);
	
	Result: 10

Flow Control – do-while / while

- **do-while**

Syntax	Example

do {
// stmt-1
} while (<condition>);
	int i = 0;
	do {
	System.out.println("In do"); i++;
	} while (i < 10);
	
	Result: Prints "In do" 11 times

- **while**

Syntax	Example

while (<condition>) {
//stmt
}
	int i = 0;
	while (i < 10) {
	System.out.println("In while"); i++;
	}
	
	Result: "In while" 10 times

Flow Control – for loop

- **for**

Syntax	Example

for (initialize; condition; expression)
{
// stmt
}
	for (int i = 0; i < 10; i++)
	{
	System.out.println(“In for”);
	}
	
	Result: Prints “In for” 10 times

For-Each Version of the for Loop

```
public class Test {
     public static void main(String args[]) {
       int evenArray[] = {2,4,6,8,10};
       int sum = 0;
       for(int i :  evenArray) {
           sum = sum + i;
       }

       System.out.println("sum::"+ sum);

   }
```

break Statement

- The break statement is used to break from an enclosing do, while, for, or switch statement.
- It is a compile error to use break anywhere else.
- 'break' breaks the loop without executing the rest of the statements in the block.

```
public class Test {

    public static void main(String[] args) {
      int i = 0;
      while (true) {
          System.out.println(i);
          i++;
          if (i > 3) {
              break;
          }
      }
    }
}
```

continue Statement

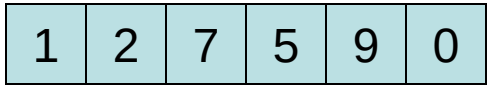
- The continue statement stops the execution of the current iteration and causes control to begin with the next iteration.
- For example, the following code prints the number 0 to 9, except 5.

```
public class Test {

  public static void main(String[] args) {
    for (int i = 0; i < 10; i++) {
      if (i == 5) {
        continue;
      }
      System.out.println(i);
    }
  }

}
```

Arrays and Strings – Arrays Declaration/Construction/Initialization

- **Array Declaration**

```
int myArray[];
int[] myArray;
double[][] doubleArray;
```

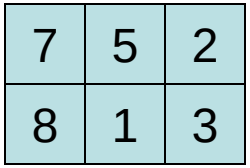


- **Array Construction**

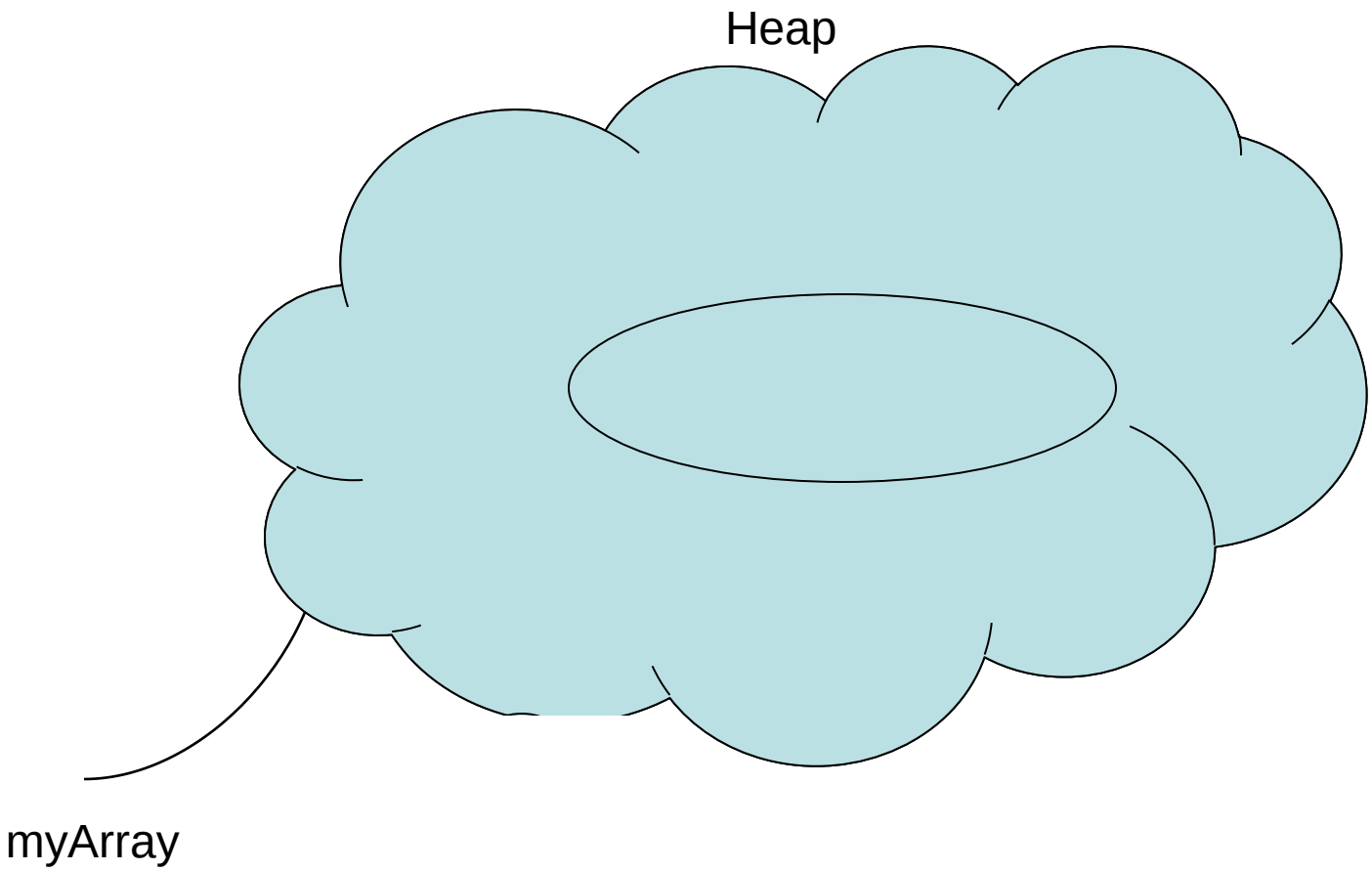
```
int[] myArray = new int[5];
int[][] twoDimArray = new int[5][4]
```

- **Array Initialization**

```
int[] myArray = new int[5];
for (int I = 0; I < 5; i++) {
   myArray[i] = i++;
}

int[5] myArray = {1,2,3,4,5};
```

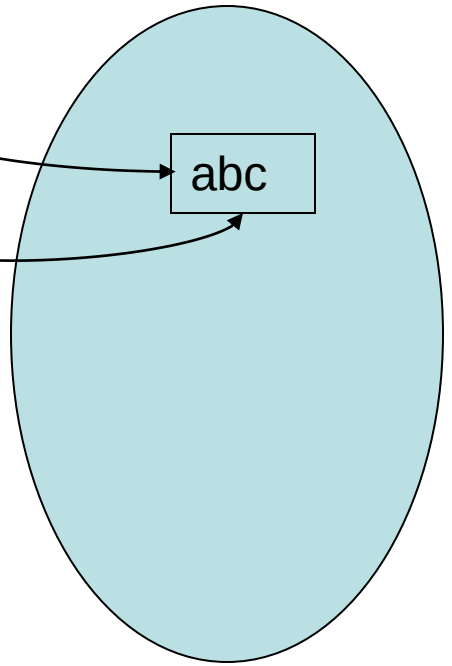


Arrays and Strings – Arrays Representation



- **Creating String Objects**

String myStr1 = new String("abc");
String myStr2 = "abc";



String Constant Pool

- **Most frequently used String methods**

  - charAt (int index)
  - compareTo (String str2)
  - concat (String str2)
  - equals (String str2)
  - indexOf (int ch)
  - length()
  - replace (char oldChar, char newChar)
  - substring (int beginIndex, int endIndex)