GUJARAT TECHNOLOGICAL UNIVERSITY

MASTER OF COMPUTER APPLICATIONS

SEMESTER: III

Subject : **4639302**

Programming in JAVA

UNIT – 1 : Introduction to Java

Programming Platform, Java Buzzwords, overview of applet and internet, History of Java, common misconception about Java, The Java Programming Environment: installing JDK, using the command line tools, using IDE, Programming Structures in Java: data types, operators, working with Strings, working with Scanner class for input and output, control flow, big number and Arrays.

What is JAVA?

- Java is a programing language that is exclusively object oriented developed by Sun Microsystem
- Object-oriented programing (OOP) is an unusual but powerful way to develop software
- In OOP, a computer program is considered to be a group of objects that interact with each other

Java as a Programming Platform

- Java is a well-designed programming language.
- Java is a platform:
  - Good programming language.
  - Efficient and secure execution environment.
  - Huge library.
- That combination has made it irresistible for over 20 years.

Java Buzzwords / Feature

1. Simple
2. Object-Oriented
3. Distributed
4. Robust
5. Secure
6. Architecture-neutral
7. Portable
8. Interpreted
9. High-performance
10. Multithreaded
11. Dynamic

1. Simple

- Java is small and simple language
- Many feature of C and C++ that are either redundant or source of unreliable code are not the part of Java
- To make java familiar to programmer, it was modeled on C and C++ languages
- Java Code "looks like a C++" code

2. Object oriented

- Object oriented though out - no coding outside of class definitions, including main()
- An extensive class library available in the core language packages

3. Distributed

- *Java has an extensive library of routines for coping with TCP/IP protocols like HTTP and FTP.*
- *Java applications can open and access objects across the Net via URLs with the same ease as when accessing a local file system.*

4. Robust

- Allows to implement robust application as comes up with exception handling, strong type checking, built in data types, garbage collection

5. Secure

- *Java is intended to be used in networked / distributed environments.*
- *Toward that end, a lot of emphasis has been placed on security.*
- *Java enables the construction of virus-free, tamper-free systems.*

6. Architecture-Neutral

- *The compiler generates an architecture-neutral object file format—the compiled code is executable on many processors, given the presence of the Java runtime system.*
- *The Java compiler does this by generating bytecode instructions which have nothing to do with a particular computer architecture.*
- *Rather, they are designed to be both easy to interpret on any machine and easily translated into native machine code on the fly.*

Portable

- *Unlike C and C++, there are no "implementation-dependent" aspects of the specification.*
- *The sizes of the primitive data types are specified, as is the behavior of arithmetic on them.*

Interpreted

- Compiler / Interpreter Combo
  - Code is complied to byte code that are interpreted by a Java Virtual Machine
  - This provides portability to any machine for which a virtual machine has been written.
  - The two step of compilation and interpretation allow for extensive code checking and improved security

High-performance

- *While the performance of interpreted bytecodes is usually more than adequate, there are situations where higher performance is required.*
- *The bytecodes can be translated on the fly (at runtime) into machine code for the particular CPU the application is running on.*

Multithreaded

- Lightweight processes, called threads, can easily be created to perform multiprocessing

Dynamic

- Java is capable of dynamically linking in new class libraries, methods and objects.
- Java can also determine the type of class though a query, making it possible to either dynamically link or abort the program, depending on the response

Overview of applet and internet

- Flash back to 1995.
- Browsers were new and served static web pages.
- Java applets made them *alive*.
- Applet code is downloaded from the internet.
- Runs safely in the “sandbox”.
- Unfortunately, the initial excitement turned into frustration.
  - Netscape and Internet Explorer used different Java versions.
  - Flash provided dynamic effects in browsers.
  - Sun and Oracle were slow to react to security breaches.
- Nowadays, Java is used in servers and smart phones.

History of Java

- 1991: James Gosling worked on “Project Green”, a system for consumer devices.
- He designed a programming language, originally called “Oak”.
- That name was trademarked, so it was renamed to “Java”.
- 1992: The first project was released, a TV switchbox called “*7”.
- Nobody cared, and the project was renamed “First Person, Inc.”
- 1994: Still nobody cared, and Gosling realized that they could build a “really cool browser...architecture-neutral, real-time, reliable, secure.”
- 1995: HotJava was released.
- 1996: Java 1.0 was released.
- 1998: Java grows up with Java 2 release. SE, ME, EE edtions.
- 2014: Java 8 has major new language features.
- 2016: Java is again the #1 language in the TIOBE ratings.

Common misconception about Java

- **Java is an extension of HTML or XML.**
  - ❑ Java is a programming language.
- **Java is an easy programming language to learn.**
  - ❑ No programming language as powerful as Java is easy.
- **Java will become a universal programming language for all platforms.**
  - ❑ This is possible in theory, but today, Java is most commonly used in backend systems and Android applications.
- **Java is just another programming language.**
  - ❑ Java has a support system that far exceeds that of most other languages.
- **Java is proprietary, and it should therefore be avoided.**
  - ❑ Java is open source.

Common misconception about Java

- **Java is interpreted, so it is too slow for serious applications.**
  - The just-in-time compiler can produce code that is as fast as C++, and sometimes faster.
- **All Java programs run inside a web page.**
  - Applets run in web pages, but most Java programs run on servers or mobile/embedded devices.
- **Java programs are a major security risk.**
  - Applet security risks are real, but in general Java is very secure.
- **JavaScript is a simpler version of Java.**
  - JavaScript was named after Java for marketing reasons.
- **With Java, I can replace my desktop computer with a cheap "Internet appliance."**
  - That was an expectation 20 years ago, and it may be true today if your Android smart phone or tablet has replaced your desktop.

The Java Programming Environment:

installing JDK

- **Download the Java Software Development Kit (JDK) from**

  http://www.oracle.com/technetwork/java/javase/downloads

  - ❑ Navigating the Oracle site requires mastery of some Java jargon.
  - ❑ Also download and unzip the documentation of the application programming interface (API).
- **Install the JDK, using the provided installer.**
- **Set the PATH environment variable.**

Java Jargon

Java Development Kit	JDK	The software for programmers who want to write Java programs
Java Runtime Environment	JRE	The software for consumers who want to run Java programs
Standard Edition	SE	The Java platform for use on desktops and simple server applications
Enterprise Edition	EE	The Java platform for complex server applications
Micro Edition	ME	The Java platform for use on cell phones and other small devices
Update	u	Oracle’s term for a bug fix release
NetBeans	—	Oracle’s integrated development environment

Using the Command-Line Tools

- Open a terminal window or command prompt
- Change to the directory where you have java classes / java code
- **cd corejava**
- **javac HelloWorld.java**
- **java HelloWorld**

using IDE

- It is useful to know how to compile programs from the command line.
- For day-to-day work, integrated development environments are more convenient.
- Excellent choices are the freely available Eclipse, NetBeans, and IntelliJ IDEA.
- ***We use command line and Notepad++***

Programming Structures in Java

- Java is case sensitive: Main ≠ main.
- Keywords public, static, etc.
- Braces { } are used for blocks.
- Statements end in semicolons.
- Everything is inside a class.
- By convention, class names are CamelCase.
- Source file must be called HelloWorld.java

Creating an Application in Java

```
/**first java code**/
class HelloWorld
{
  public static void main(String[] args)
  {
     System.out.println(“JVIMS");
  }
}
```

Compile : javac HelloWorld.java

Run : java HelloWorld

Document Section
package Statements
import Statements
interface Statements
Class Definitions
main() Definition

data types

Integers

Four integer types:

int	**4 bytes**	**–2,147,483,648 to 2,147,483, 647 (just over 2 billion)**
short	**2 bytes**	**–32,768 to 32,767**
long	**8 bytes**	**–9,223,372,036,854,775,808 to 9,223,372,036,854,775,807**
byte	**1 byte**	**–128 to 127**

Literals:

Long:	4000000000L
Hex:	0xCAFE
Binary:	0b1111_0100_0010_0100_0000

Floating-Point Numbers

- Two floating-point types:
  - **float**
    - 4 bytes
    - Approximately ±3.40282347E+38F
    - (6–7 significant decimal digits)
  - **double**
    - 8 bytes
    - Approximately ±1.79769313486231570E+308
    - (15 significant decimal digits)
- **float literals: 0.5F**
- **Only use float if a library requires it.**
- **Special values**
- **Double.POSITIVE_INFINITY, Double.NEGATIVE_INFINITY, Double.NaN.**

The char Type

- Originally used to describe Unicode characters.
- Nowadays: “Code Units” in the UTF-16 encoding.
- Every Unicode character requires one or two char values:
  - A has “code point” U+0041 and is encoded as a single char value (hex 0041 or decimal 65).
  - 𝕆 has “code point” U+1D546 and is encoded by two code units with hex values D835 DD46.
- Avoid char unless you know that you won't run into Unicode characters ≥ U+10000.
- Character literals enclosed in single quotes:
  - 'A', '\n', '\u2122'.

The boolean Type

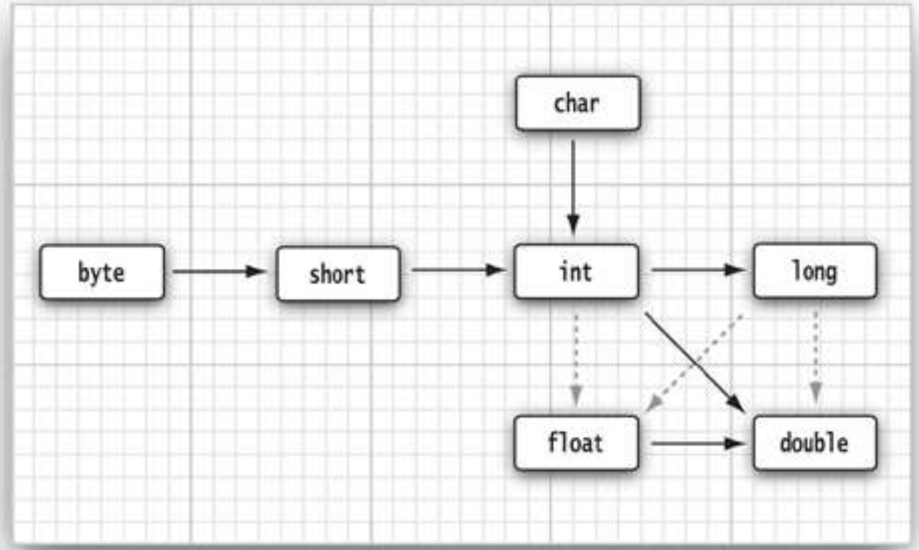
- Two values: false, true.
- No conversion between int and boolean.
- All together, Java has eight primitive types:
  - int, long, short, byte
  - double, float
  - char
  - boolean

Mathematical Operators

- Arithmetic: +, -, *, /
- Integer division and modulus: / and % (with integer operands):
  - 15 / 2 is 7.
  - 15 % 2 is 1.
  - 15.0 / 2 is 7.5.

Type Conversions

Automatic type conversions



$$\begin{tikzcd}
 & & char \arrow[d] & \\
byte \arrow[r] & short \arrow[r] & int \arrow[r] \arrow[d, dashed] \arrow[dr] & long \arrow[d, dashed] \arrow[dl, dashed] \\
 & & float \arrow[r] & double
\end{tikzcd}$$

(Dotted arrows indicate possible precision loss.)

explicit conversions:

- double x = 9.997;
- int nx = (int) x;
- int rx = (int) Math.round(x)

More Operators

- Combining assignment with operators:
  - n += 4; // Same as n = n + 4
  - Also -=, *=, /=, %=, and so on.
- Increment, decrement:
  - n++; n--;
- Relational operators:
  - ==, !=, <, <=, >, >=
- boolean operators:
  - &&, ||, !
- Bitwise operators:
  - &, |, ^, ~, >>, >>>, <<
- Conditional operator:
  - x < y ? x : y

Enumerated Types

- **A type with a restricted set of values.**
- **Example:**

```
enum Size { SMALL, MEDIUM, LARGE, EXTRA_LARGE };
```

- **Can declare variable of this type:**

```
Size s = Size.MEDIUM;
```

- **Variable of type s can only hold size values or null.**

working with Strings

- Sequences of Unicode characters.
- String literals enclosed in double quotes:
  - "Java"
- Instances of String class.
- Use substring method to extract substrings:
  - String greeting = "Hello";
  - String s = greeting.substring(0, 3);
- Positions start at zero.

Concatenation (+) joins strings:

- String s1 = “abc";
    - String s2 = “xyz";
    - String message = s1 + s2;
- If one operand is not a string, it is turned into a string:
    - int age = 13;
    - String rating = "PG" + age;

More About Strings

- String comparison:
  - "Hello".equals(greeting)
  - "Hello".equalsIgnoreCase(greeting)
- Caution: Do not use the == operator.
  - "Hello".substring(0, 3) == "Hel"
- Empty string "" has length 0.
- null indicates no string at all.

The String API

- Many other useful String methods.
- **trim** gives a new string, trimming leading and trailing white space.
- **toLowerCase** gives a new string that coverts all uppercase characters to lowercase.
- **indexOf, lastIndexOf** find the location of a substring.
- Check out the online API documentation!

working with Scanner class for input and output

- **Read console input from a Scanner:**
  - Scanner in = new Scanner(System.in);
- **Use nextLine, next, nextInt, nextDouble to read input:**
  - int age = in.nextInt();
- **Need to add import statement:**
  - import java.util.*;

Formatted Output

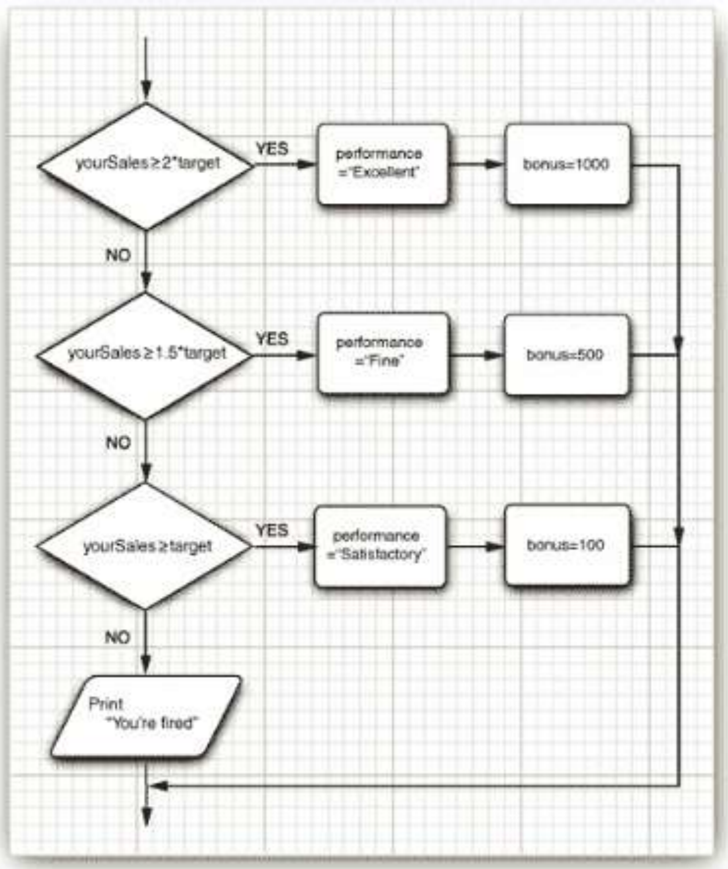
- **Use printf for formatted printing:**
  - System.out.printf("Price:%8.2f", 10000.0 / 3.0);
  - Prints Price: 3333.33

- **Use f for floating-point, d for integer, s for strings and other objects.**
- **Flags modify output:**
  - System.out.printf("%(,.2f", -10000.0 / 3.0);
  - prints (3,333.33)

- **Use String.format if you don't want to print:**
  - String message = String.format("Hello, %s. Next year, you'll be %d", name, age);

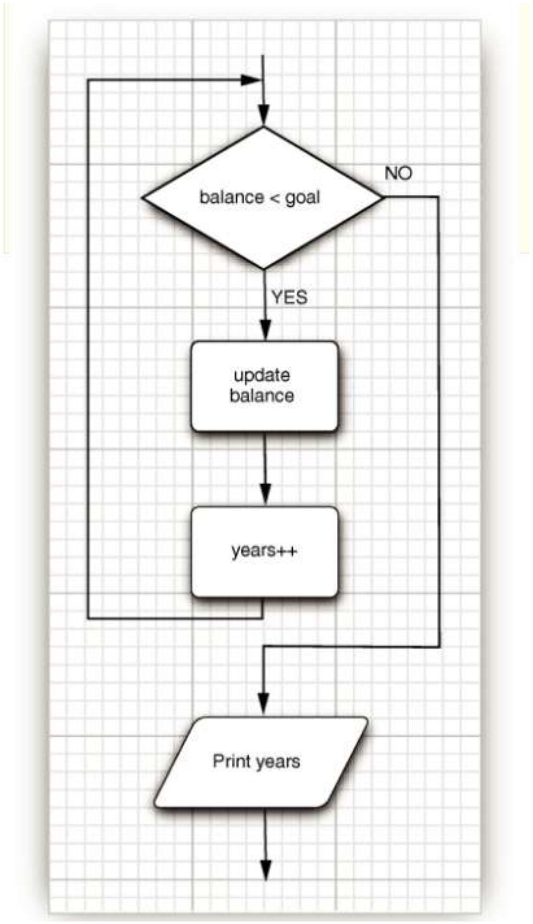
control flow

The if Statement



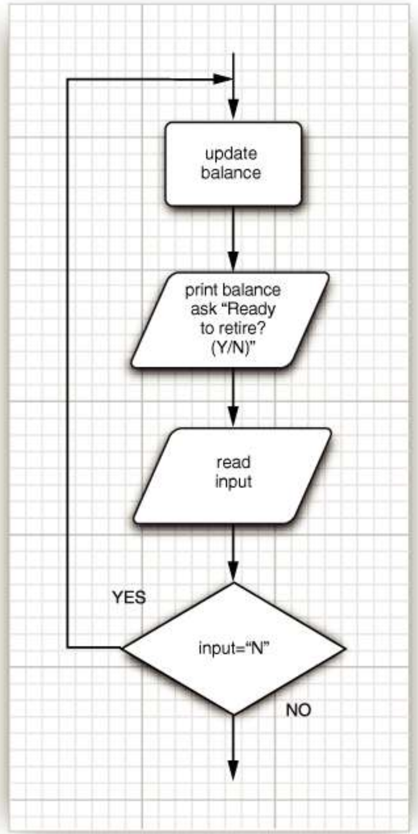
```
if (yourSales >= 2 * target)
{
   performance = "Excellent";
   bonus = 1000;
}
else if (yourSales >= 1.5 * target)
{
   performance = "Fine";
   bonus = 500;
}
else if (yourSales >= target)
{
   performance = "Satisfactory";
   bonus = 100;
}
else
{
   System.out.println("You're fired");
}
```

The while Statement



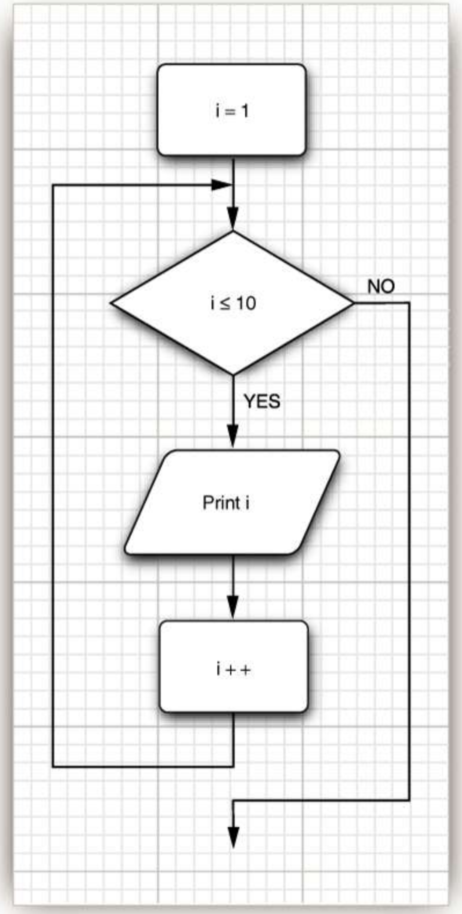
```
while (balance < goal)
{
   balance += payment;
   double interest = balance * interestRate / 100;
   balance += interest;
   years++;
}
System.out.println(years + " years.");
```

The do/while Statement



```
do
{
   balance += payment;
   double interest = balance * interestRate / 100;
   balance += interest;
   year++;
   // print current balance
   . . .
   // ask if ready to retire and get input
   . . .
}
while (input.equals("N"));
```

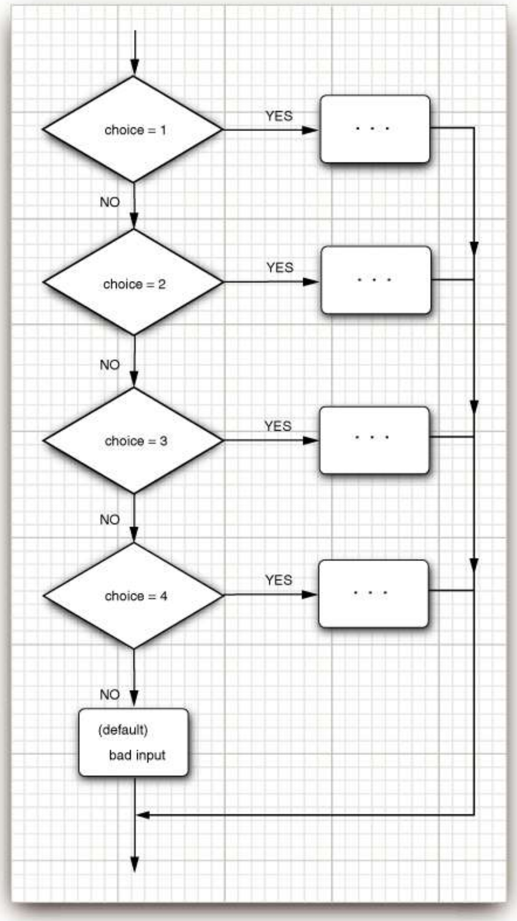
The for Statement



```
for (int i = 1; i <= 10; i++)
{
                System.out.println(i);

}
```

The switch Statement



```
Scanner in = new Scanner(System.in);
System.out.print("Select an option (1, 2, 3, 4) ");
int choice = in.nextInt();
switch (choice) {
   case 1:
      . . .
      break;
   case 2:
      . . .
      break;
   case 3:
      . . .
      break;
   case 4:
      . . .
      break;
   default:
      // bad input
      . . .
      break;
}
```

The break Statement

- Breaks out of a loop:

```
 i=0;
 while (i <= 100)
 {
      i ++;
      if (i >= 10) {
          break;
      }
     System.out.print(i);
 }
```

- A “labeled break” lets you break out of multiple nested loops.

The continue Statement

```
i=0;
while (i <= 100)
{
    i ++;
    if (i== 10)
    {
       continue;
    }
   System.out.print(i);
}
```

- Continues with the next loop iteration:

big numbers

- If the precision of int and double doesn't suffice, use BigInteger or BigDecimal.
- Turn an int into a BigInteger:
  - BigInteger a = BigInteger.valueOf(100);
- Use methods such as add and multiply to combine big numbers:
  - BigInteger c = a.add(b); // c = a + b
  - BigInteger d = c.multiply(b.add(BigInteger.valueOf(2))); // d = c * (b + 2)
- Example:
  - Pick 60 numbers out of 490.
  - Your odds of winning are 1 in 716395843461995557415116222540092933411717612789263493493351013459481104668848. Good luck!

Arrays

- **int[] is an array of integers.**
- **Array variable declaration:**

```
int[] a;
```

- **new operator creates array:**

```
int[] a = new int[100];
```

- **Array indexes are from 0 to a.length - 1.**
- **Use [] to access elements:**

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

- **Or use the “for each” loop:**

```
for (int element : a)
   System.out.println(element);
```

Supplying Element Values

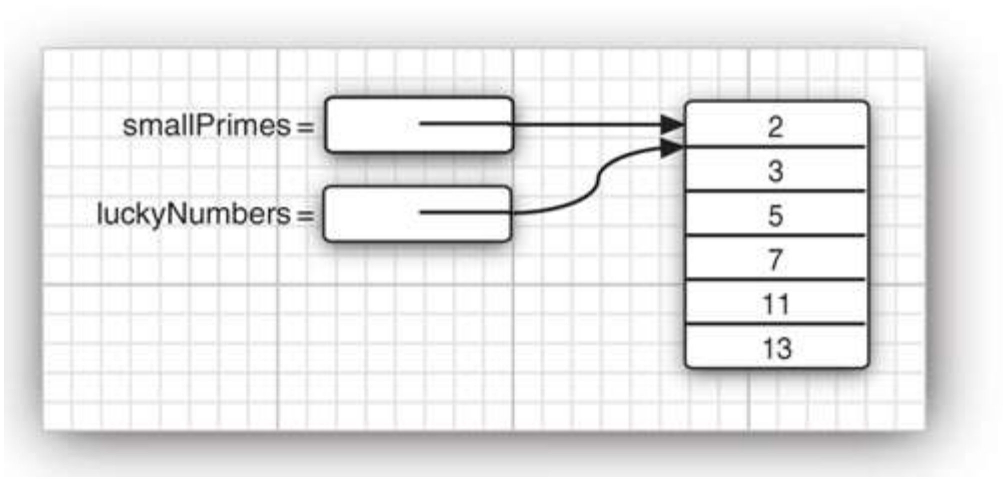
- Array initializers:

```
int[] smallPrimes = { 2, 3, 5, 7, 11, 13 };
```

- Anonymous arrays:

```
new int[] { 17, 19, 23, 29, 31, 37 }
```

Copying Arrays



Copying array variables yields two references to the same array:

```
int[] luckyNumbers = smallPrimes;
luckyNumbers[5] = 12; // now smallPrimes[5] is also 12
```

Use Arrays.copyOf to make a true copy:

```
int[] copiedLuckyNumbers =
        Arrays.copyOf(luckyNumbers, luckyNumbers.length);
```

Multidimensional Arrays

- Without initializer:

```
int[][] magicSquare = new int[ROWS][COLUMNS];
```

- With initializer:
- int[][] is an array of arrays or a two-dimensional array:

```
int[][] magicSquare = {
        { 16, 03, 02, 13 },
        { 05, 10, 11, 08 },
        { 09, 06, 07, 12 },
        { 04, 15, 14, 01 }      };
```

- Use two indexes to access element: magicSquare[1][2]
- Value is 11.

Multidimensional Arrays

- **Use this loop to traverse the elements:**

```
for (int[] row : magicSquare)
  for (int element : row)
                do something with value
```

- **If the rows have different lengths, the array is "ragged":**

```
int[][] triangle = new int[ROWS][];
for (int i = 0; i < ROWS; i++)      triangle[i] = new int[i + 1];
```

- **If the rows have fix lengths**

```
int[][] triangle = new int[ROWS][COLS];
for (int i = 0; i < ROWS; i++)
      for (int j = 0; j < COLS; j++)
```

Contact:

Bipin S. Rupadiya

Assistant Professor, JVIMS

Mo. : +91-9228582425

Email: info@bipinrupadiya.com

Blog : www.BipinRupadiya.com