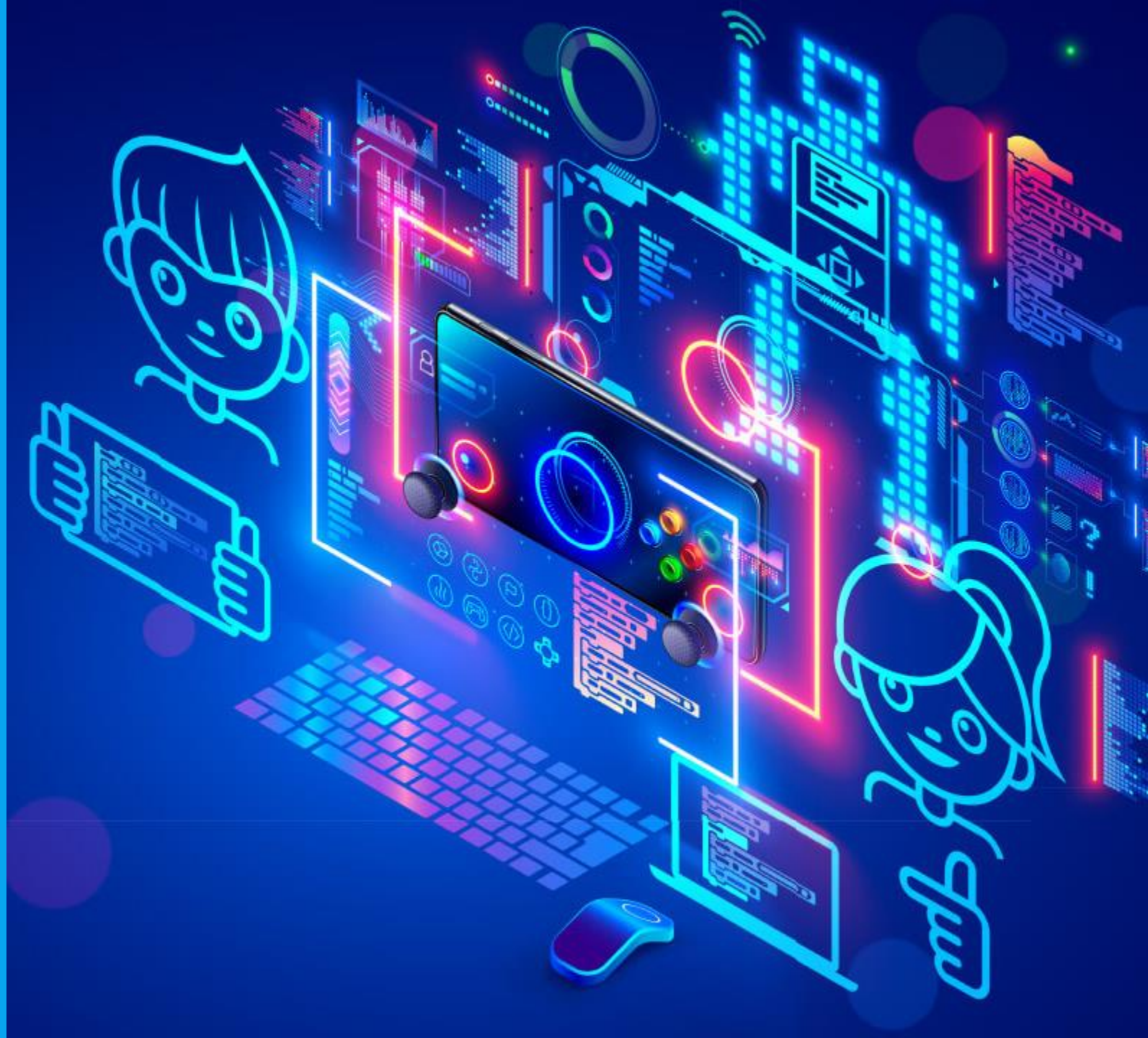# CS 24 AP Computer Science A Review
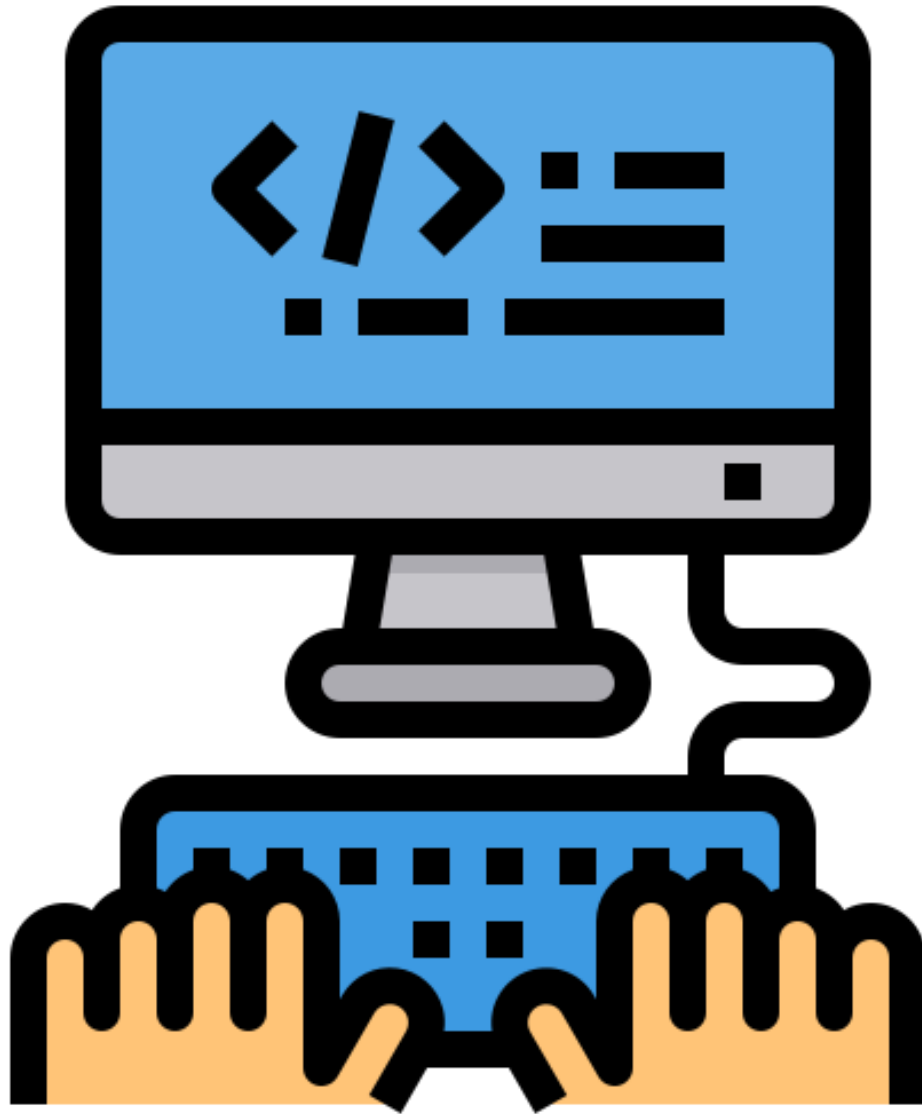
Week 1: Fundamentals of Java Language

DR. ERIC CHOU
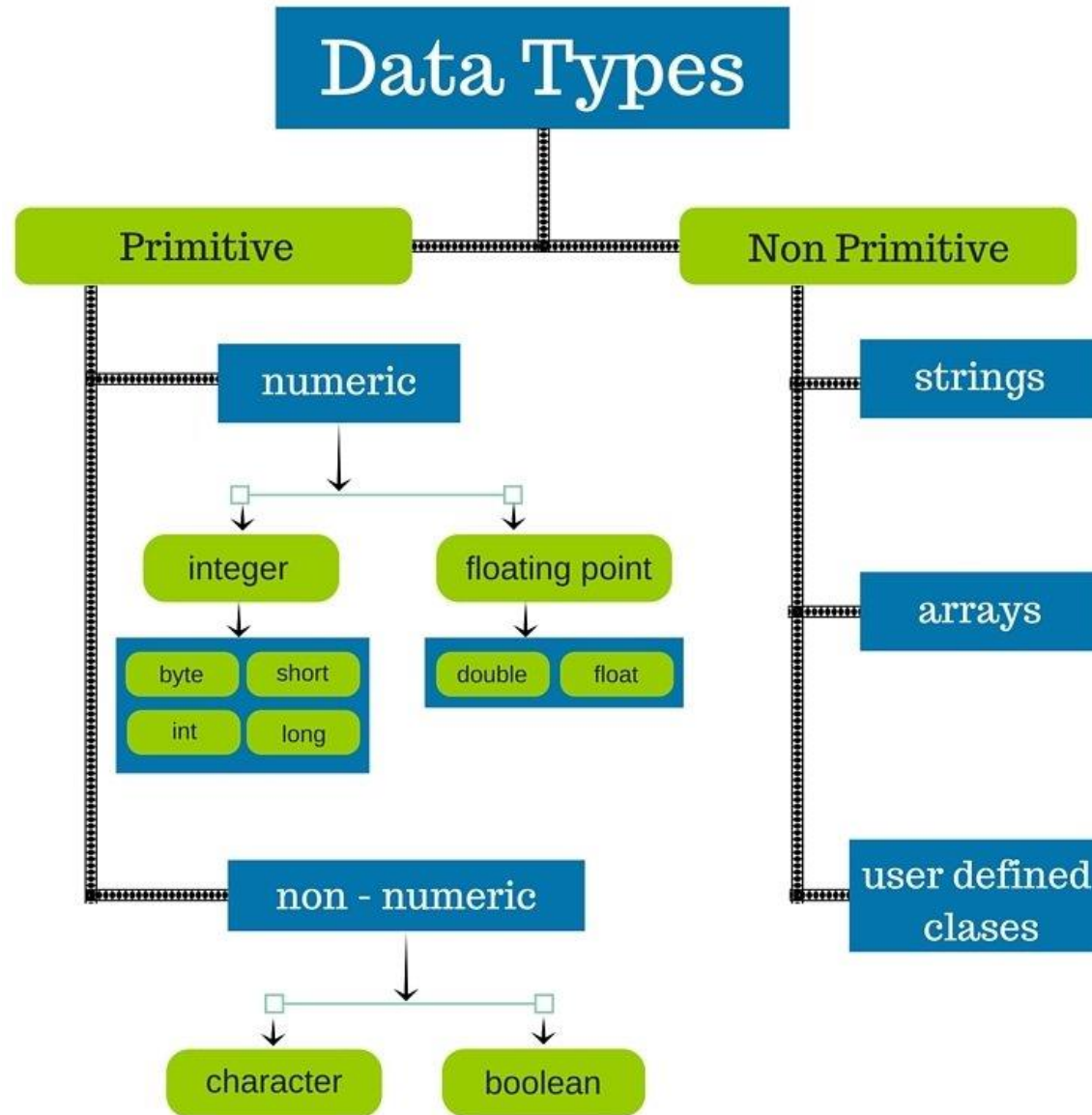IEEE SENIOR MEMBER

Topics

- Data Types

- Basic Java Programs

- Java Operators, Short-Circuits

- Input and Output

- Loops

- Number System

- Errors and Exceptions

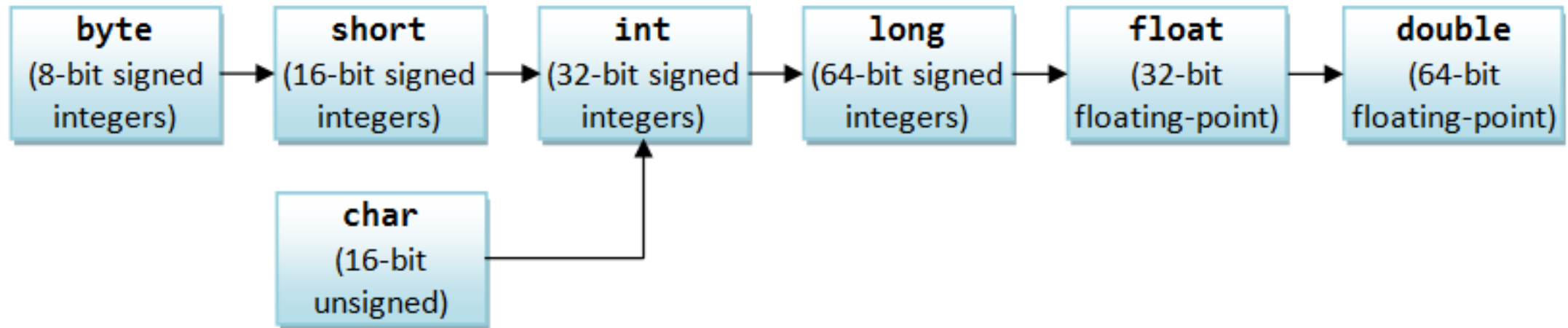# Data Types

Section 1

Java Primitive Types

Type	Size	Range	Default
boolean	1 bit	true or false	false
byte	8 bits	[-128, 127]	0
short	16 bits	[-32,768, 32,767]	0
char	16 bits	['\u0000', '\uffff'] or [0, 65535]	'\u0000'
int	32 bits	[-2,147,483,648 to 2,147,483,647]	0
long	64 bits	$[-2^{63}, 2^{63}-1]$	0
float	32 bits	32-bit IEEE 754 floating-point	0.0
double	64 bits	64-bit IEEE 754 floating-point	0.0

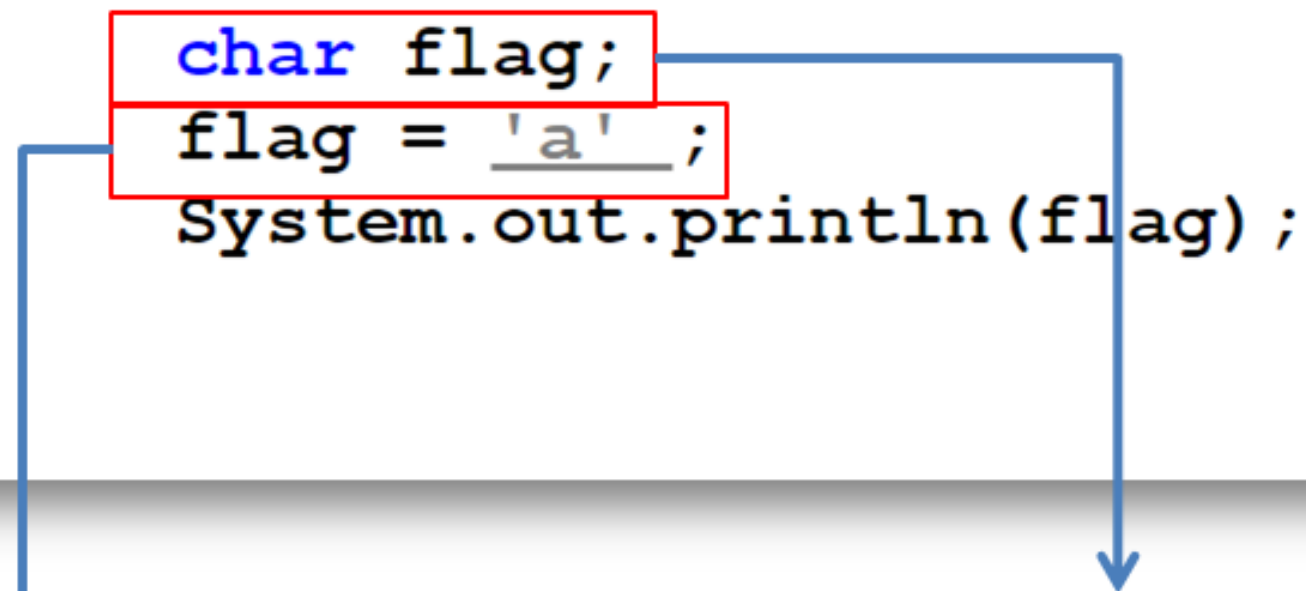# Numeric Data Types and Operations

Java has six numeric types for integers and floating-point numbers with operators +, -, *, . and %

| byte (8-bit signed integers) | short (16-bit signed integers) | int (32-bit signed integers) | long (64-bit signed integers) | float (32-bit floating-point) | double (64-bit floating-point) |

char (16-bit unsigned)

Orders of Implicit Type-Casting for Primitives

```java
// A Java program to demonstrate char data type
class CharacterExample1 {
    public static void main(String args[])
    {
        char flag;
        flag = 'a';
        System.out.println(flag);
    }
}
```
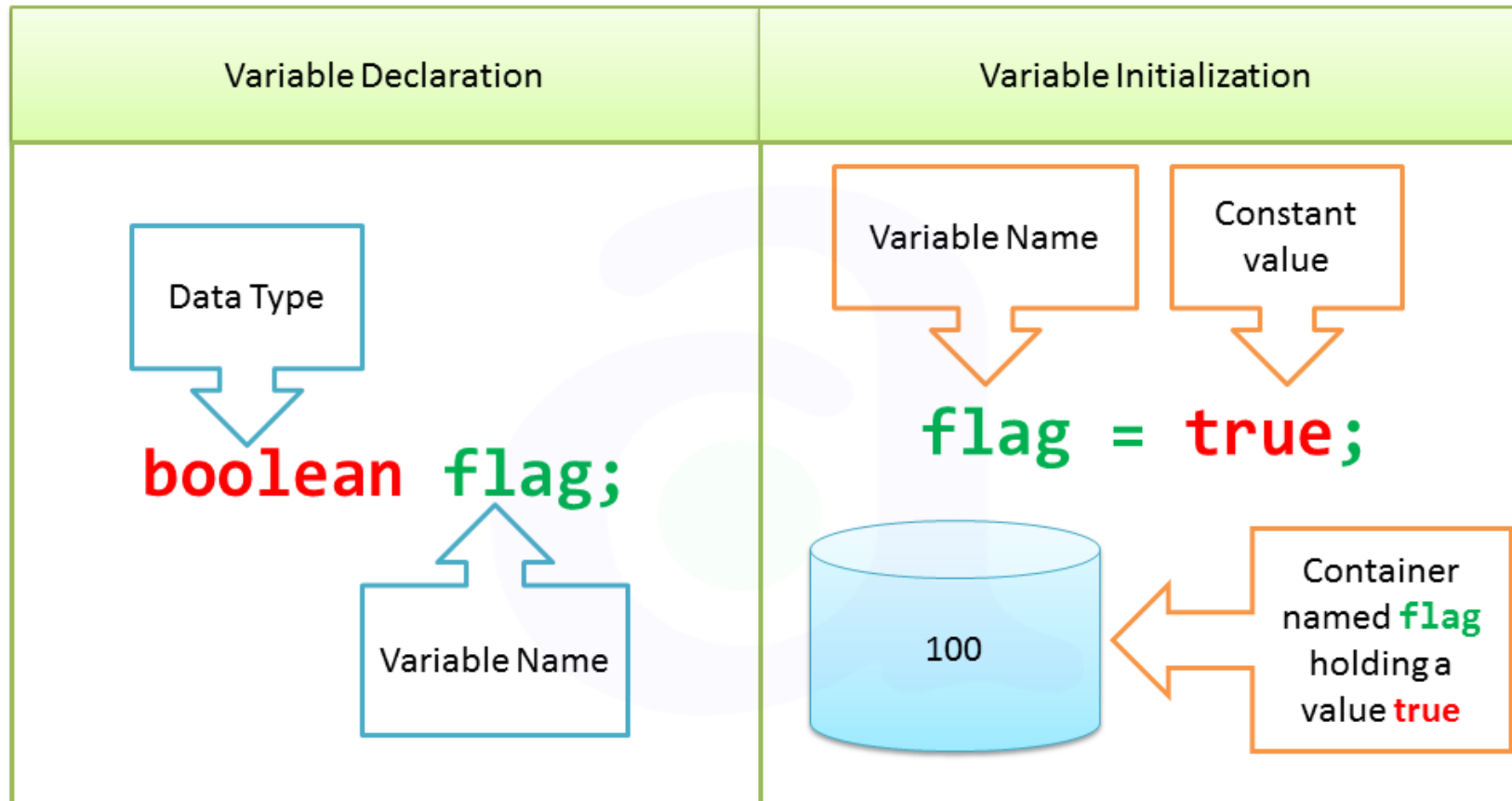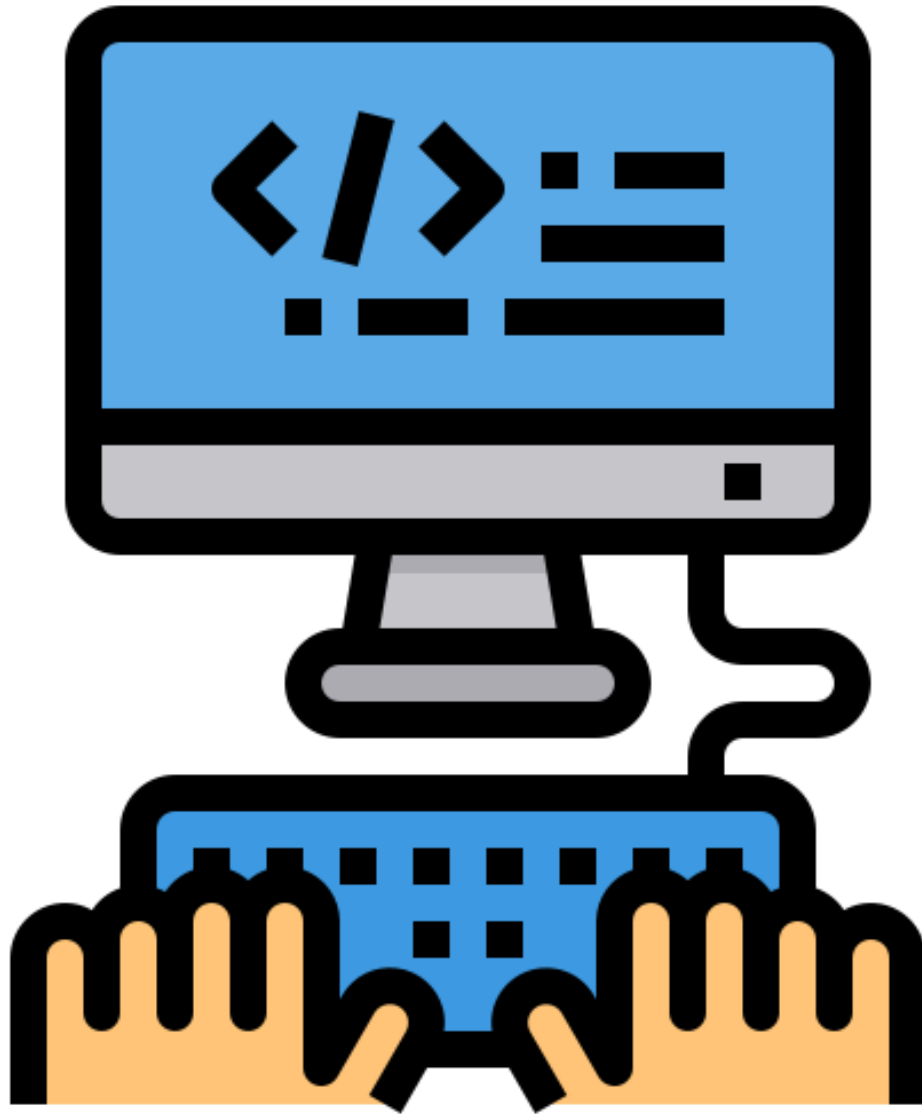
char variable Initialization

char variable Declaration

Boolean Data Type

boolean variable Declaration and Initialization

Basic Java Programs

Section 2

```
public class Charge          ← class name
{
  private double rx, ry;     ← instance variables
  private double q;

  public Charge(double x0, double y0, double q0)   ← constructor
  { rx = x0; ry = y0;  q = q0;  }

  public double potentialAt(double x, double y)
  {
      double k = 8.99E09;
      double dx = x - rx;        ← instance variable names
      double dy = y - ry;
      return k * q / Math.sqrt(dx*dx+dy*dy)
  }

  public String toString()
  { return q + ": " + "("+ rx + ", " + ry +")";}

  public static void main(String[] args)   ← test client
  {
      double x = Double.parseDouble(args[0]);
      double y = Double.parseDouble(args[1]);
      Charge c1 = new Charge(.51, .63, 21.3);   ← create and initialize object
      Charge c2 = new Charge(.13, .94, 81.9);   ← invoke constructor
      double v1 = c1.potentialAt(x, y);
      double v2 = c2.potentialAt(x, y);         ← object name / invoke method
      StdOut.println(v1+v2);
  }
}
```

instance methods

Anatomy of a class

class Class_Name{
   final static CONSTANTS;
   static methods(){}
   data fields (instance variables);
   Constructor(){}
   instanceMethods(){}
   overrideMethods(){}
   public static void main(){}
}

Java comments and code block marks

Characters	Name	Description
//	Double Slash	Line comment
/* */	Slash star and star slash	Opening and closing of comment text
/** */	Slash double-star and star-slash	Opening and closing of Javadoc comment text Javadoc comment can be extracted into HTML file using the JDK's Javadoc command (Use to describe a module, a method or a variable)
{ }	Braces	For a code block.
[ ]	brackets	For the index variable
( )	parenthesis	For the boundary of an expression or a logic conditions
" "	double quotes	For boundary of a string of text data

Variable, Class and Program

```java
public class Example {
    public static void main(String[] args){
        // Variable Declaration
        double radius = 5.0;
        // Input part
        Scanner input = new Scanner(System.in);
        radius = input.nextDouble();
        // Processing part
        double area = Math.PI * radius * radius;
        //  Output Part
        System.out.println(area);
    }
}
```

radius
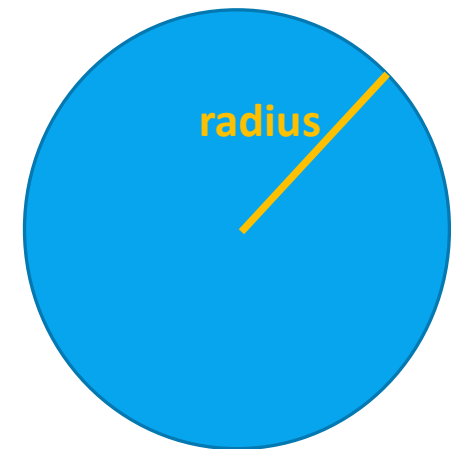
Naming Conventions

Variable and Method names:

Use lowercase for variables and methods. If a method is longer than one word, the first letter for each word, except the first word, may sometime in uppercase.

Example: int aa;

double women_age;

int functionName();

int functionToComputeInterest();

Constant names:

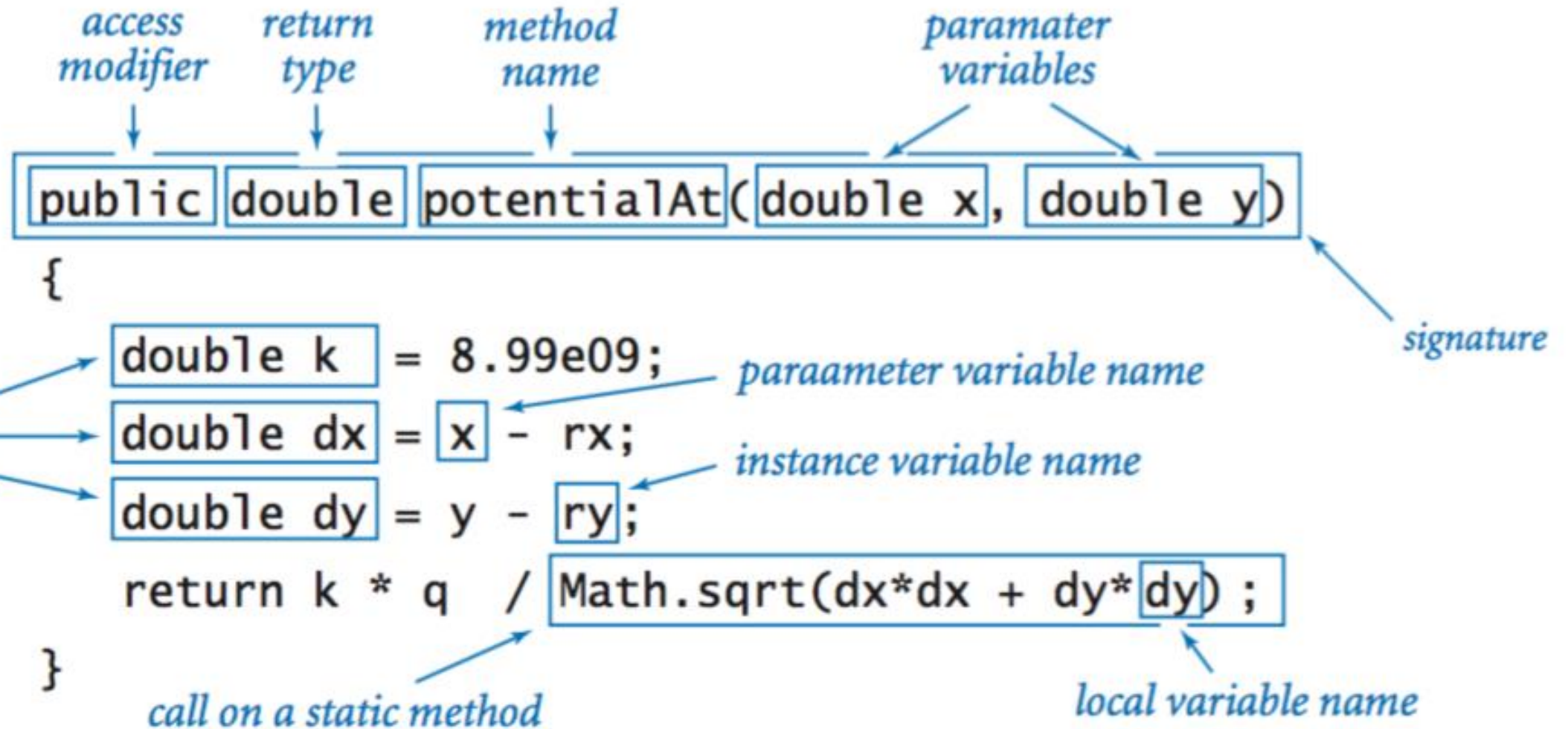Capitalize every letter in a constant and use underscore between words – for example, the constant PI and MAX_VALUE;

Learning Channel

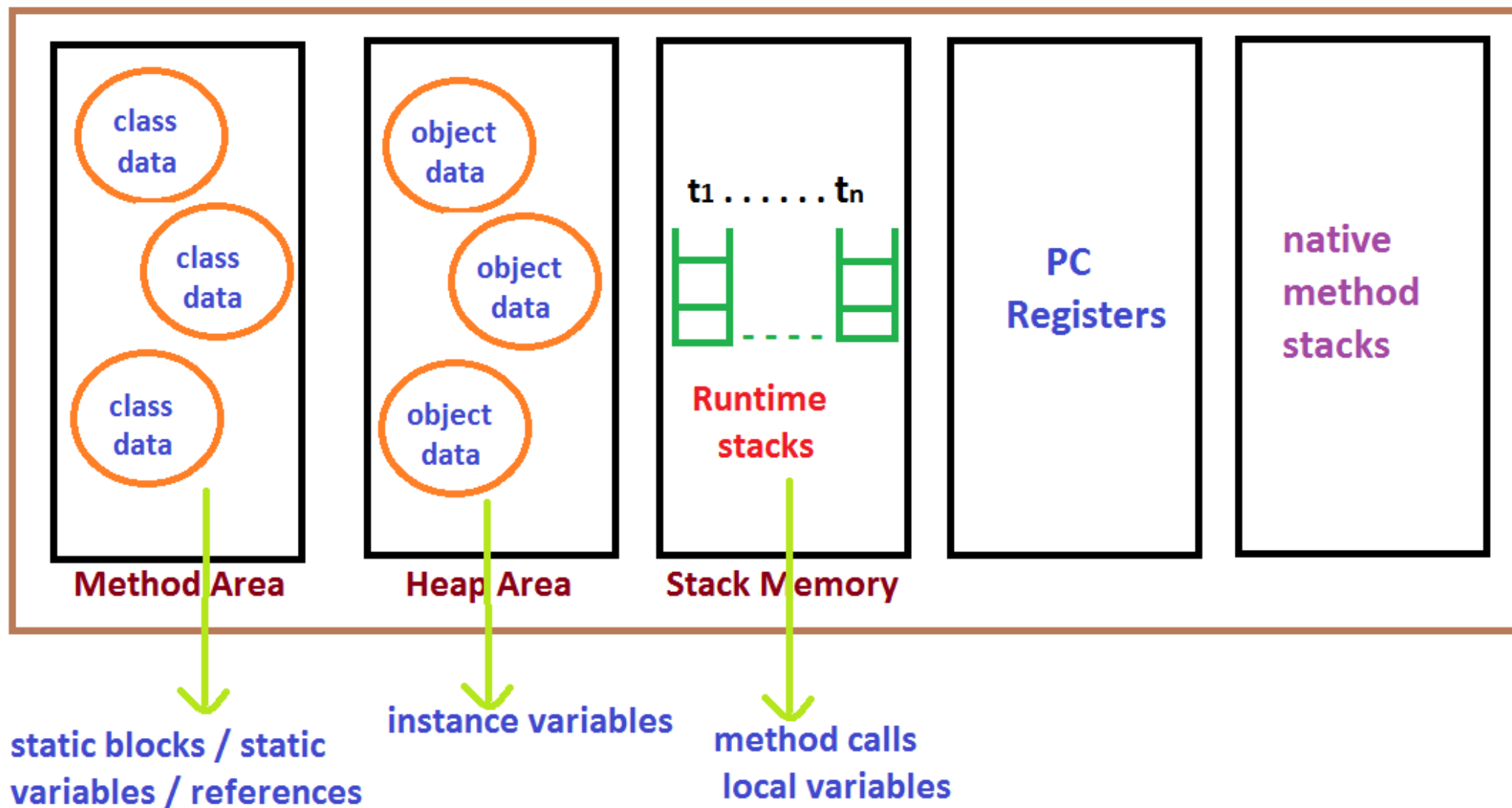# Naming Conventions
(module and package)

Class names:

- Capitalize the first letter of each word in the name.
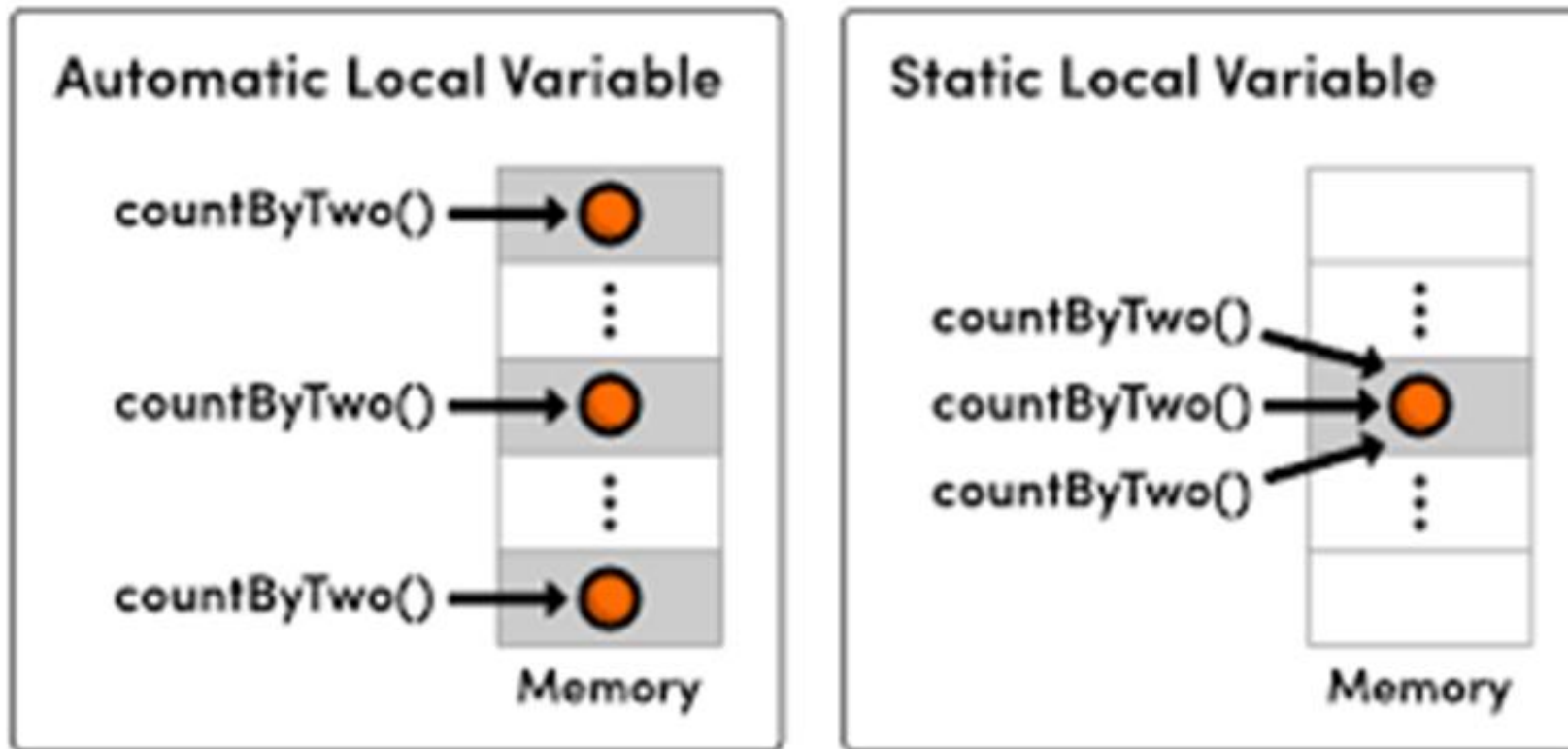- For example, the class name `ComputeArea`.

Package names:

- The whole package name in lower case.

Diagram labels: access modifier, return type, method name, paramater variables, signature, local variables, paraameter variable name, instance variable name, local variable name, call on a static method

```java
public double potentialAt(double x, double y)
{
    double k  = 8.99e09;
    double dx = x - rx;
    double dy = y - ry;
    return k * q  / Math.sqrt(dx*dx + dy*dy);
}
```

class data

class data

class data

object data

object data

object data

t1 . . . . tn

Runtime stacks

PC Registers

native method stacks

Method Area

Heap Area

Stack Memory

static blocks / static variables / references

instance variables

method calls local variables

Static Members

Static Members/Instance Members
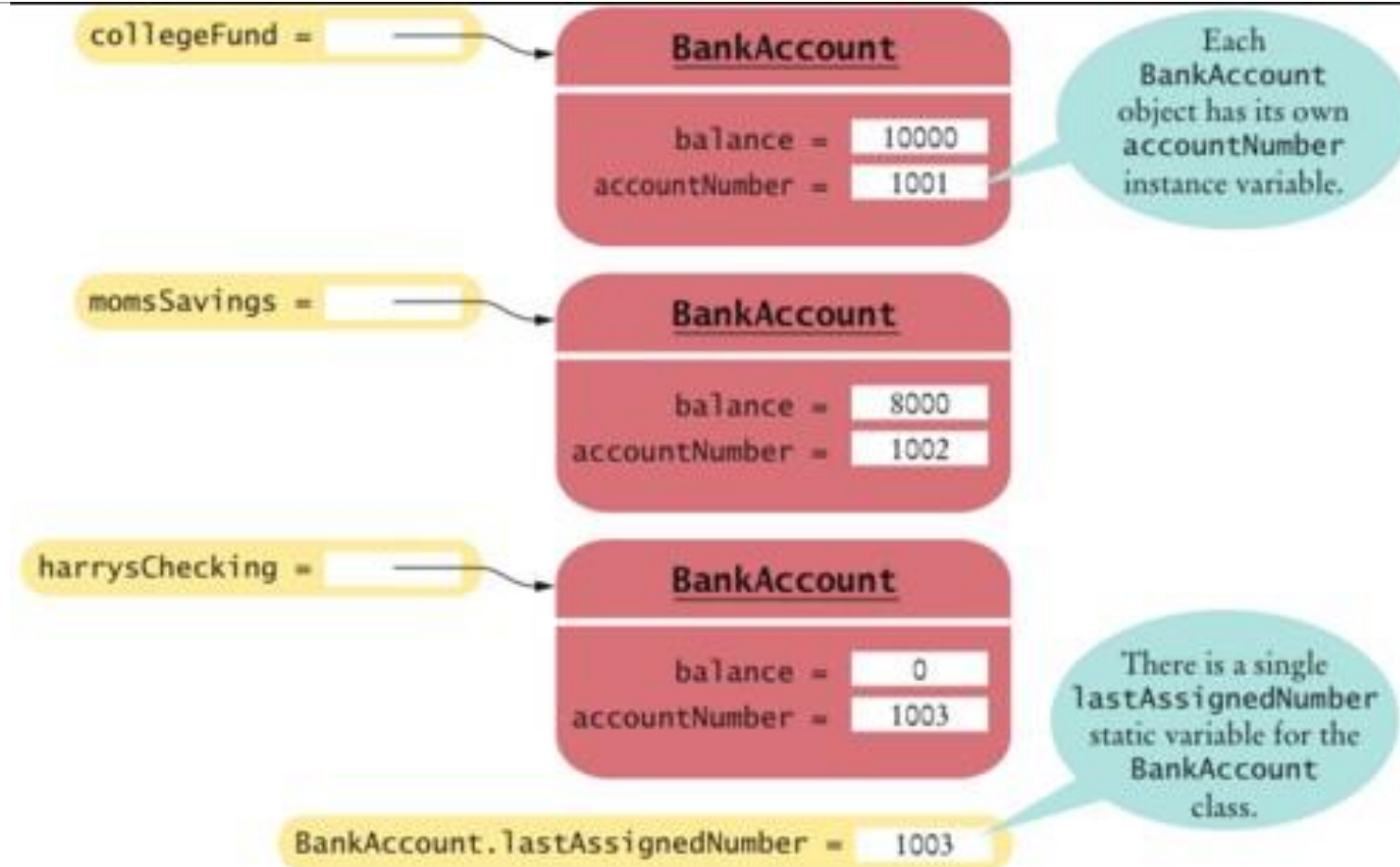
Java stack & heap memory allocation

Stack

```
public class StackRef {

    public void first(){
        second();
        //after
    }



    public void second() {
        Car c = new Car();
    }

}
```

first()

second()
first()

second() ref c
first()

Heap

Car

```
public class HeapRef{
    Car c = new Car();

    public void first() {
        c = New Car();
    }
}
```

Car
c
Ref

Car
c
Car
Ref
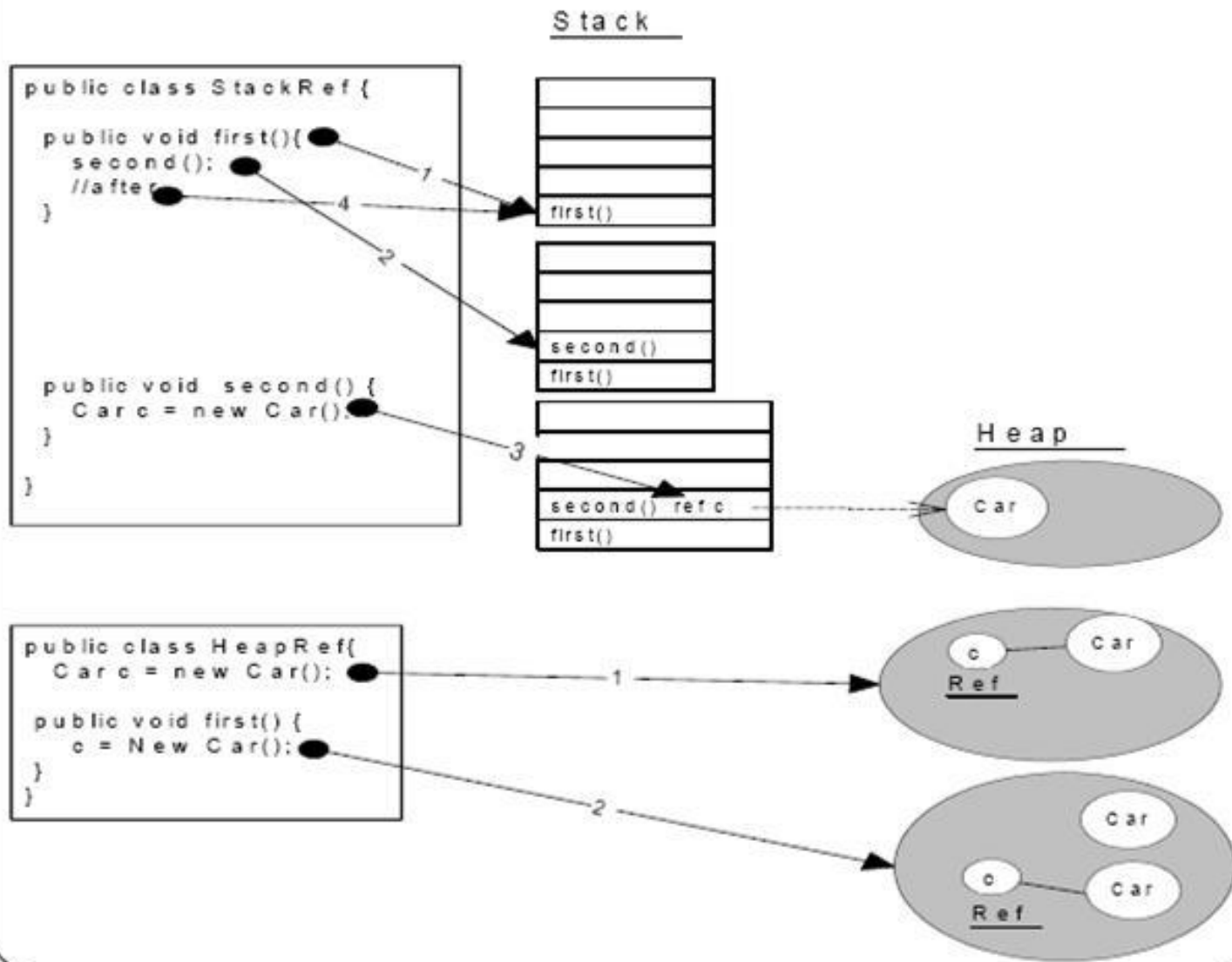
Java Operators

Section 3

Java Arithmetic Operators

Operator	Result
+	Addition
-	Subtraction (also unary minus)
*	Multiplication
/	Division
%	Modulus
++	Increment
+=	Addition assignment
-=	Subtraction assignment
*=	Multiplication assignment
/=	Division assignment
%=	Modulus assignment
--	Decrement

Integer Division and Its Usage

- int a = b * q + r;   // a, b, q, r are all integers
- q = a / b;
- b = a / q;
- r = a % q;
- r = a % b;

Multiple of N

- if (m % 2 == 0) System.out.println("m is multiple of 2");

- **Leap Year:**
  boolean leapYear = (year %400==0) || (year %100!=0 && year %4==0);

- **GCD Euclidean Algorithm:**

Week1/GCD.java

```java
8  public class GCD
9  {
10     public static int gcd(int m, int n){
11         if (m%n==0) return n;
12         return gcd(n, m%n);
13     }
14     public static void main(String[] args){
15         System.out.println(gcd(48, 32));
16         System.out.println(gcd(65, 52));
17         System.out.println(gcd(52, 65));
18     }
19  }
```

Reverse of Digits
Week1/ReverseDigits.java

- Take the least significant digit by taking modulo-10;

- Multiply-Shift and Add operations

```java
public class ReverseDigits
{
    public static int reverseDigit(int x){
        int r=0;
        int d=0;
        while (x>0){
            d = x % 10;
            r = r * 10 + d;
            x /= 10;
        }
        return r;
    }
    public static void main(String[] args){
        System.out.print("\f");
        System.out.println(reverseDigit(12345));
    }
}
```

Convert Decimal to Binary String

Week1/ToBinary.java

```java
public class ToBinary
{
    public static String toBinary(int dec){
        String r = "";
        while (dec >0){
            int bit = dec % 2;
            r = ""+bit+r;
            dec /=2;
        }
        return r;
    }
    public static void main(String[] args){
        System.out.println(toBinary(127));      1111111
        System.out.println(toBinary('A'));       1000001
        System.out.println(toBinary('z'));       1111010
    }
}
```
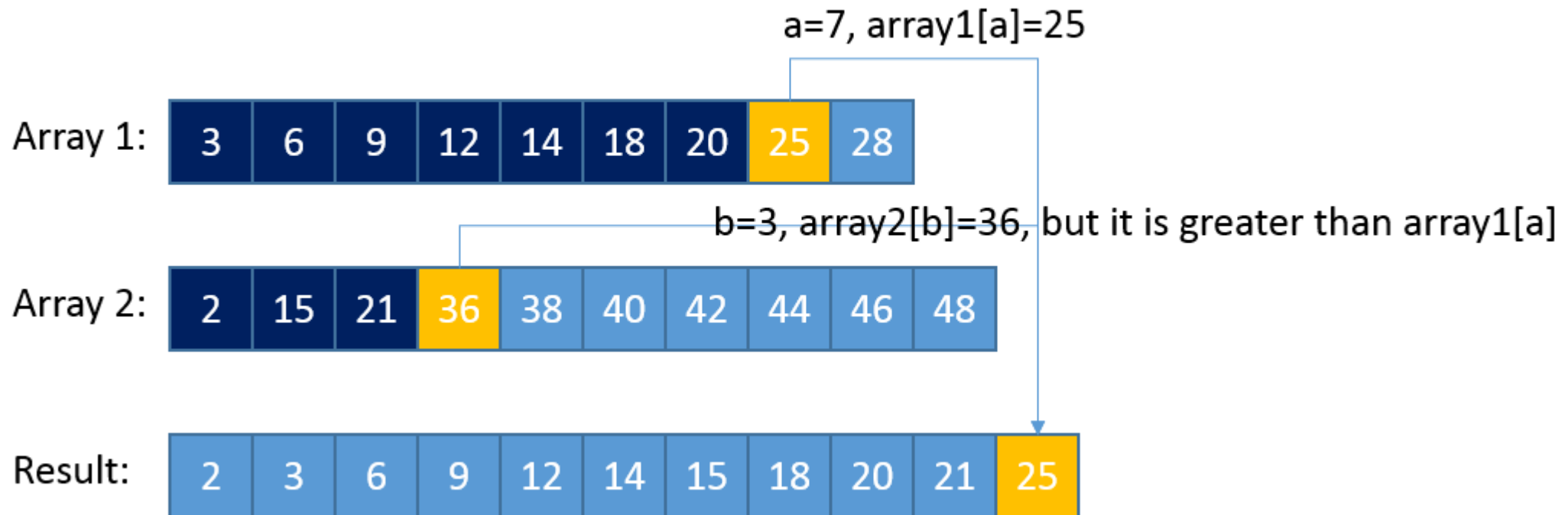
Merge of Arrays
Week1/MergeArray.java

1. Use integer as indexing pointer to an array. (p1, p2, p3)

2. Post-increment operator

3. Alternating increment of pointers

4. Deal with remaining elements
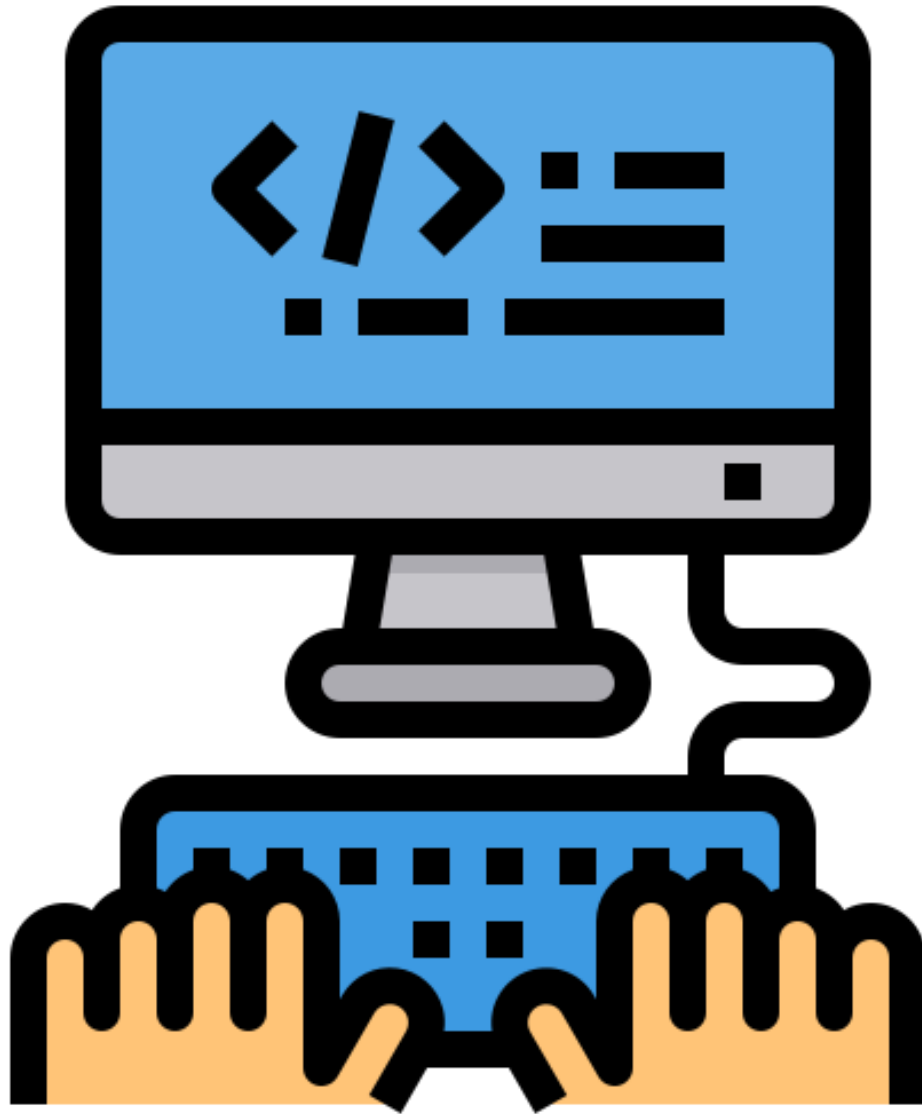
5. Transcopy

Merge Two Arrays

Merge

a=7, array1[a]=25

Array 1: | 3 | 6 | 9 | 12 | 14 | 18 | 20 | 25 | 28 |

b=3, array2[b]=36, but it is greater than array1[a]

Array 2: | 2 | 15 | 21 | 36 | 38 | 40 | 42 | 44 | 46 | 48 |

Result: | 2 | 3 | 6 | 9 | 12 | 14 | 15 | 18 | 20 | 21 | 25 |

```java
import java.util.Arrays;
public class MergeArray
{
    public static int[] mergeArray(int[] a1, int[] a2){
        int[] a3 = new int[a1.length+a2.length];
        int p1=0, p2=0, p3=0;
        while (p1<a1.length && p2<a2.length){
            if (a1[p1]<=a2[p2]) a3[p3++]=a1[p1++];
                        else a3[p3++]=a2[p2++];
         }
        while (p1<a1.length){a3[p3++] = a1[p1++];}
        while (p2<a2.length){a3[p3++] = a2[p2++];}

        return a3;
    }
    public static void main(String[] args){
        int[] ary1 = {1, 3, 3, 5, 7, 9, 12, 23};
        int[] ary2 = {2, 2, 4, 8, 10, 14, 16, 18};
        System.out.println(Arrays.toString(ary1));
        System.out.println(Arrays.toString(ary2));
        System.out.println(Arrays.toString(mergeArray(ary1, ary2)));
    }
}
```

```
[1, 3, 3, 5, 7, 9, 12, 23]
[2, 2, 4, 8, 10, 14, 16, 18]
[1, 2, 2, 3, 3, 4, 5, 7, 8, 9, 10, 12, 14, 16, 18, 23]
```

Relational Operators

Section 4

Boolean Data Type

The Boolean data type declares a variable with the value either true or false.

Relational Operators				
Java Operator	Math Symbol	Name	Example	Result
<	<	Less than	radius < 0	false
<=	≤	Less than or Equal to	radius <= 0	false
>	>	Greater than	radius > 0	true
>=	≥	Greater than or equal to	radius >= 0	true
==	=	Equal to	radius == 0	false
!=	≠	Not Equal to	radius != 0	true

Boolean literals: **true** and **false.** These are the only values that will be returned by the Boolean expressions.
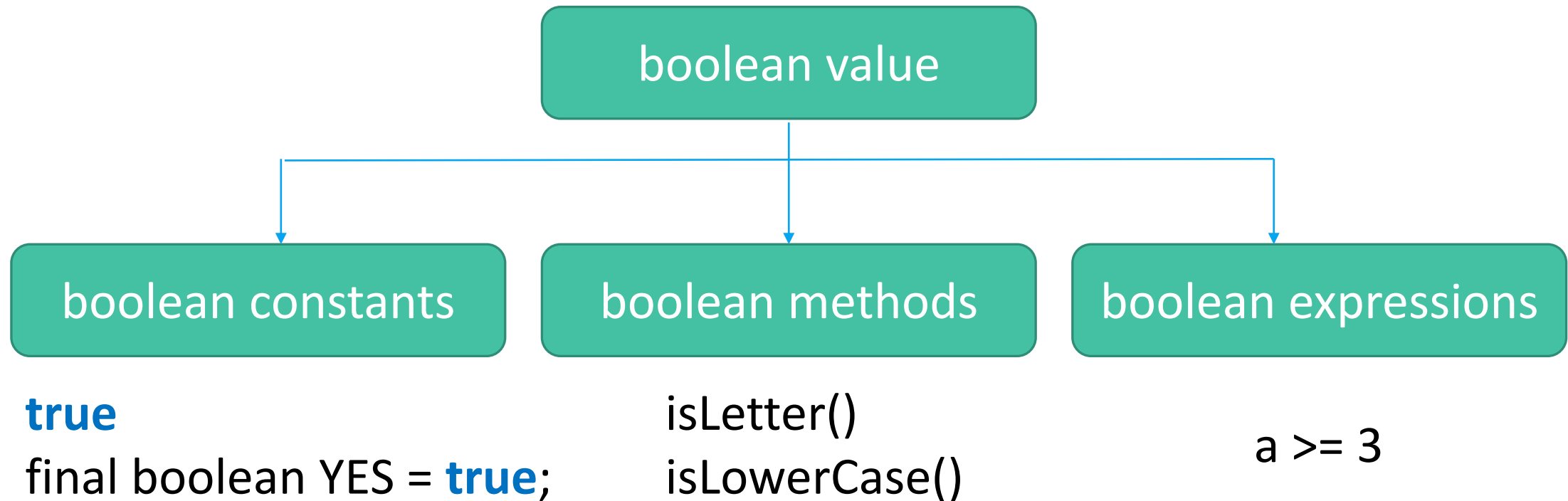
Java boolean Expressions

For example, suppose you have declared two variables: **int i = 5;   int j = 10;**

Expression	Value	Explanation
`i == 5`	`true`	The value of `i` is `5`.
`i == 10`	`false`	The value of `i` is not `10`.
`i == j`	`false`	`i` is `5`, and `j` is `10`, so they are not equal.
`i == j - 5`	`true`	`i` is `5`, and `j - 5` is `5`.
`i > 1`	`true`	`i` is `5`, which is greater than `1`.
`j == i * 2`	`true`	`j` is `10`, and `i` is `5`, so `i * 2` is also `10`.

boolean values

```
boolean value
```

```
boolean constants    boolean methods    boolean expressions
```

true
final boolean YES = **true**;

isLetter()
isLowerCase()

a >= 3

Logical Operators for Implementation of Boolean Logic

Boolean Operators		
Operator	Name	Description
!	not	Logical negation
&&	and	Logical conjunction
||	or	Logical disjunction
^	exclusive or	Logical exclusion (non-AP)

INPUTS		OUTPUTS					
A	B	AND	NAND	OR	NOR	EXOR	EXNOR
0	0	0	1	0	1	0	1
0	1	0	1	1	0	1	0
1	0	0	1	1	0	1	0
1	1	1	0	1	0	0	1

Truth Table for Operator !

p	!p	Example (assume age = 24, gender = 'M')
true	false	!(age > 18) is false, because (age > 18) is true.
false	true	!(gender != 'M') is true, because (grade != 'M') is false.

Truth Table for Operator &&

p1	p2	p1 && p2	Example (assume age = 24, gender = 'F')
false	false	false	(age > 18) && (gender == 'F') is true, because (age > 18) and (gender == 'F') are both true.
false	true	false	
true	false	false	(age > 18) && (gender != 'F') is false, because (gender != 'F') is false.
true	true	true	

Truth Table for Operator ||

p1	p2	p1 || p2	Example (assume age = 24, gender = 'F')
false	false	false	(age > 34) || (gender == 'F') is true, because (gender == 'F') is true.
false	true	true	
true	false	true	(age > 34) || (gender == 'M') is false, because (age > 34) and (gender == 'M') are both false.
true	true	true	

Truth Table for Operator ^

p1	p2	p1 ^ p2	Example (assume age = 24, gender = 'F')
false	false	false	(age > 34) ^ (gender == 'F') is true, because (age > 34) is false but (gender == 'F') is true.
false	true	true	
true	false	true	(age > 34) || (gender == 'M') is false, because (age > 34) and (gender == 'M') are both false.
true	true	false	

Short-Circuit Evaluation
Week1/ShortCircuit.java
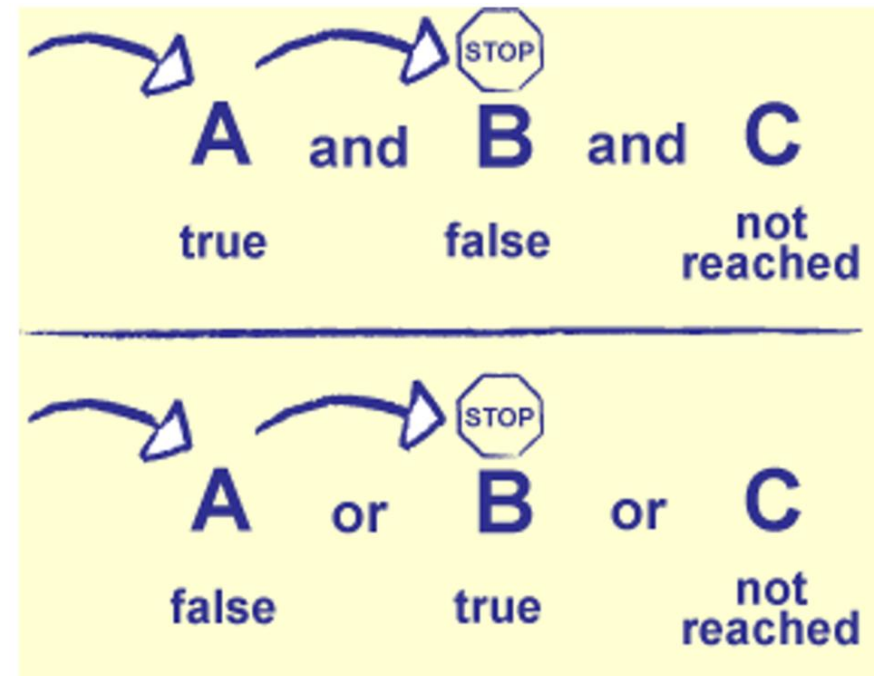
```java
public class ShortCircuit{
  public static void main(String[] argv) {
    int denom = 0;
    int num = 3;
    if (denom != 0 && num / denom > 10) {
      System.out.println("Here");
    } else {
      System.out.println("There");
    }
  }
}
```

Short-Circuit Example:
int count =0;
int total = 0;
boolean result = (count !=0 && **total/count >0**);

A and B and C
true   false   not reached

A or B or C
false   true   not reached

Name	AND form	OR form
Identity law	$1A = A$	$0 + A = A$
Null law	$0A = 0$	$1 + A = 1$
Idempotent law	$AA = A$	$A + A = A$
Inverse law	$A\overline{A} = 0$	$A + \overline{A} = 1$
Commutative law	$AB = BA$	$A + B = B + A$
Associative law	$(AB)C = A(BC)$	$(A + B) + C = A + (B + C)$
Distributive law	$A + BC = (A + B)(A + C)$	$A(B + C) = AB + AC$
Absorption law	$A(A + B) = A$	$A + AB = A$
De Morgan's law	$\overline{AB} = \overline{A} + \overline{B}$	$\overline{A + B} = \overline{A}\overline{B}$

Conditional Statement (Non-AP Subset)

Longer Array Length:

```
int len = a.length > b.length ? a.length : b.length;
```

If statement

3.b) If false

3.a) If true

1.   **2.**

if (condition)

4.   {

   // body of the if
   // statements to be executed

   }

5. // statements outside the if

Test Expression → true → Body of if → Statement just below if; false path bypasses body
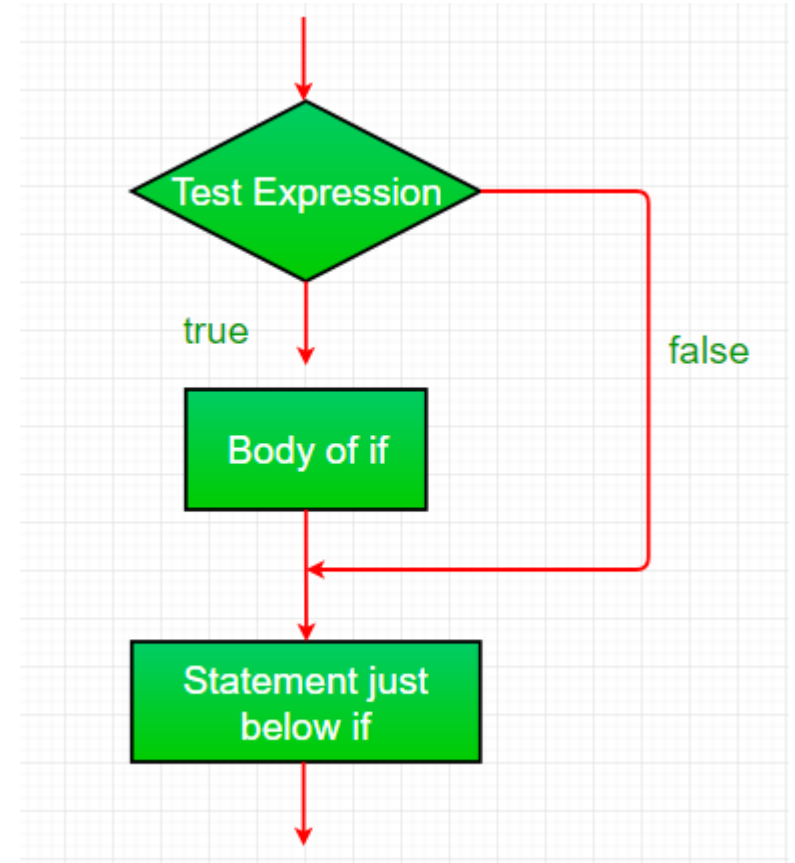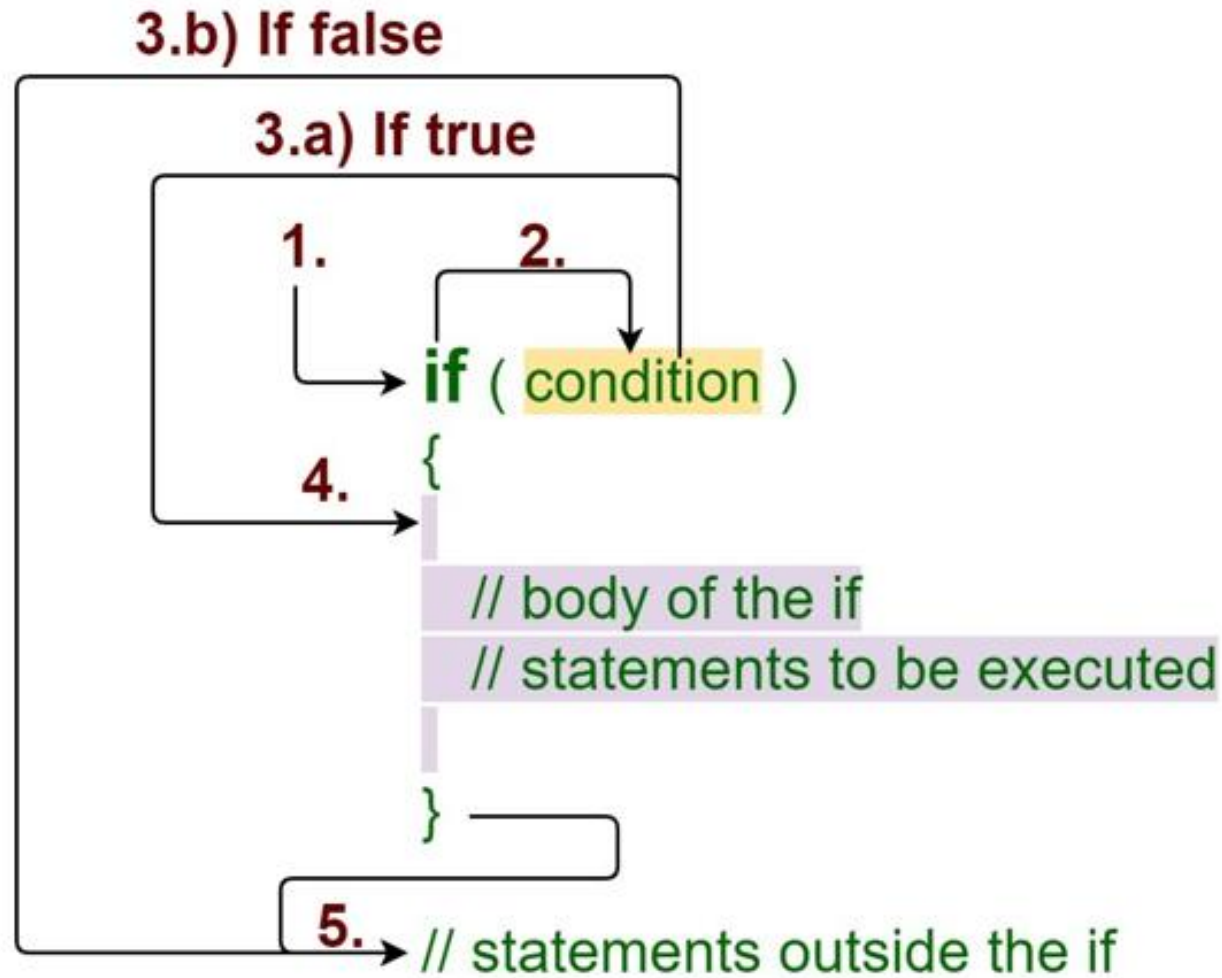
If-Statements and If-Then-Else Statements

Standard If-Else Statement:
int m=3;
boolean even;
if (m%2==0) even = true;
else even = false;

Standard If-Else Statement:
int m=3;
boolean odd;
if (m%2!=0) odd = true;
else odd = false;

Default True:
int m=3;
boolean even = true;
if (m%2!=0) even = false;

Default False:
int m=3;
boolean odd = false;
if (m%2!=0) odd = true;

Conditional Statement: (Non-AP)
int m=3;
boolean odd = (m%2!=0) true: false;
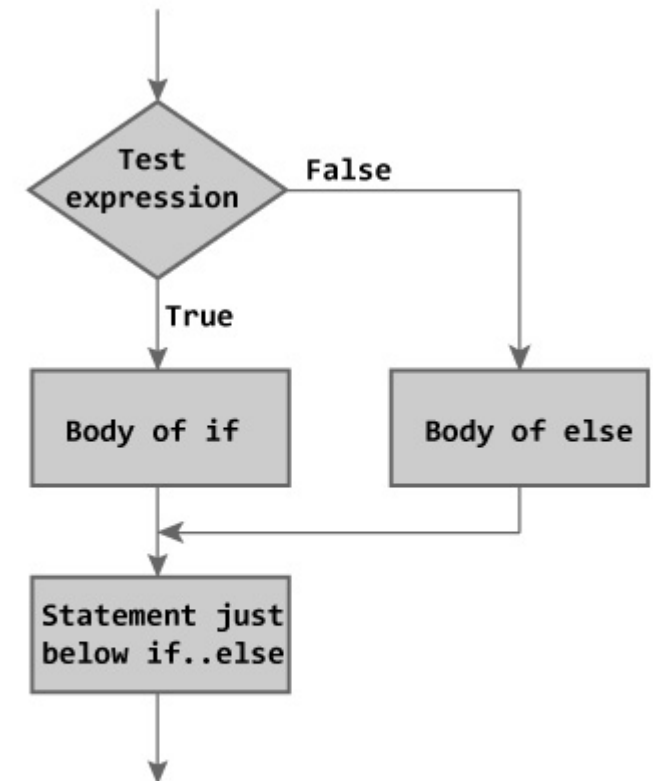
Figure: Flowchart of if...else Statement

Dangling If
else belongs to closest if

The <u>else</u> clause matches the most recent <u>if</u> clause in the same block.

```
int i = 1;
int j = 2;
int k = 3;

if (i > j)
   if (i > k)
      System.out.println("A");
else
      System.out.println("B");
```
(a)

Equivalent

```
int i = 1;
int j = 2;
int k = 3;

if (i > j)
   if (i > k)
      System.out.println("A");
   else
      System.out.println("B");
```
(b)

Learning Channel
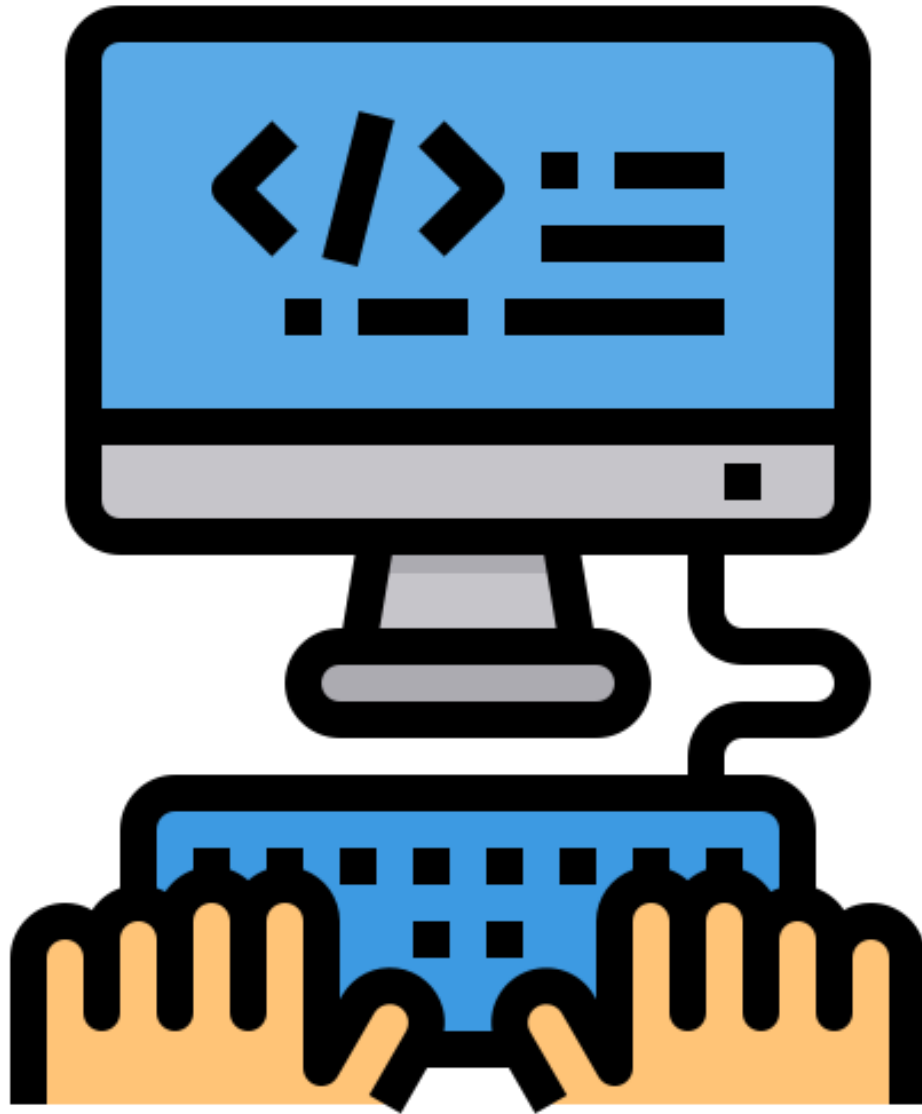
Input and Output

Section 5

Input/Output

- Usually not tested.

- Sometimes, the problem statement will have input functions, you just consider the function as the problem statement said,

- e.g.

```
int a = readInt();
```
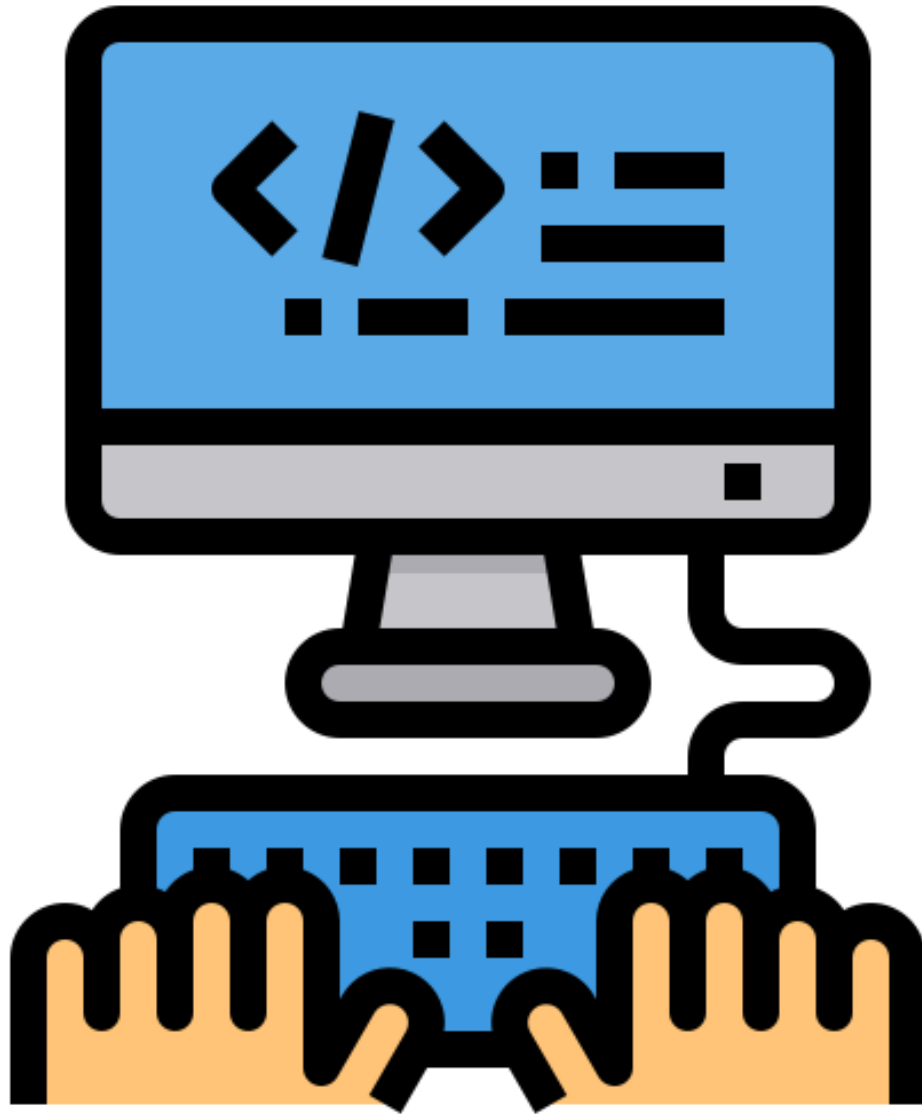
print() and println()

- Print with new line mark,

- Print without new line mark.

- printf() usually not tested.

Escape Sequences

Constant	Meaning
\t	Insert a tab in the text at this point.
\b	Insert a backspace in the text at this point.
\n	Insert a newline in the text at this point.
\r	Insert a carriage return in the text at this point.
\f	Insert a formatted in the text at this point.
\'	Insert a single quote character in the text at this point.
\"	Insert a double quote character in the text at this point.
\\	Insert a backslash character in the text at this point.

Loops

Section 6

Loops for Repetition

```
System.out.println("Welcome to Java!");
System.out.println("Welcome to Java!");
System.out.println("Welcome to Java!");
System.out.println("Welcome to Java!");
System.out.println("Welcome to Java!");
System.out.println("Welcome to Java!");
…
…
…
System.out.println("Welcome to Java!");
System.out.println("Welcome to Java!");
System.out.println("Welcome to Java!");
```

100 times

Solution to it: while-loop

```java
int count = 0;
while (count < 100) {
    System.out.println("Welcome to Java");
    count++;
}
```

while loop

The syntax for the while loop is:

while (loop-continuation-condition) {

   // loop body

   Statement(s);

}

Flow Diagram of a while loop
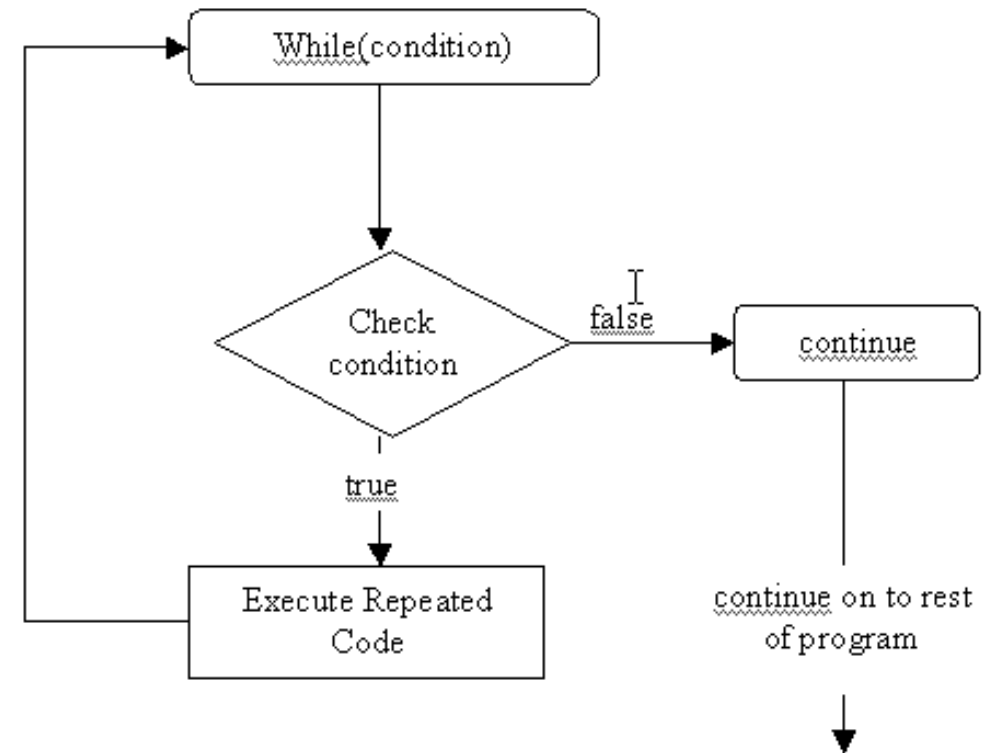
Comparison of if-statement and while-loop

```java
int x = 0;                      // if-statement
if (x < 10) {
    System.out.println("Welcome to Java.");
}

int x = 0;                      // while-loop
while (x < 10) {
    System.out.println("Welcome to Java.");
    x++;
}
```

LOOP Structures Supported By Java

Loops:

- for-loop (later lecture)

- while-loop

- do-while-loop

- for-each-loop

Loop Breaks: (later in other lecture)

- {}   /* empty braces as pass function */

- Continue /* skip the rest of iteration */

- Break   /* skip the rest of loop */

- Return   /* skip the rest of function */

- System.exit(0); /* skip the rest of program */

for loop

Syntax of for loop:

for (initial_condition; continuation-condition; action-after-each-iteration)
{

  // loop body:

  Statement(s);

}

Declaring and Initializing
loop control variable

Checking
condition

Incrementing loop
control variable

for (int i =0; i<10 ; i++) {

  // Loop statements to be executed

}

Trace for Loop

```
int i;
for (i = 0; i < 2; i++) {
  System.out.println(
     "Welcome to Java!");
}
```

Declare i

Trace for Loop

```
int i;
for (i = 0; i < 2; i++) {
  System.out.println(
    "Welcome to Java!");
}
```

Execute initializer
i is now 0

Trace for Loop

```
int i;
for (i = 0; i < 2; i++) {
  System.out.println( "Welcome to Java!");
}
```

(i < 2) is true
since i is 0

Trace for Loop

```
int i;
for (i = 0; i < 2; i++) {
    System.out.println("Welcome to Java!");
}
```

Print Welcome to Java

Trace for Loop

Execute adjustment statement
i now is 1

```
int i;
for (i = 0; i < 2; i++) {
    System.out.println("Welcome to Java!");
}
```

Trace for Loop

```
int i;
for (i = 0; i < 2; i++) {
  System.out.println("Welcome to Java!");
}
```

(i < 2) is still true
since i is 1

Trace for Loop

```java
int i;
for (i = 0; i < 2; i++) {
    System.out.println("Welcome to Java!");
}
```

Print Welcome to Java

Trace for Loop

```
int i;
for (i = 0; i < 2; i++) {
  System.out.println("Welcome to Java!");
}
```

Trace for Loop

```java
int i;
for (i = 0; i < 2; i++) {
    System.out.println("Welcome to Java!");
}
```

(i < 2) is false
since i is 2

Trace for Loop

```
int i;
for (i = 0; i < 2; i++) {
    System.out.println("Welcome to Java!");
}
```

Exit the loop. Execute the next statement after the loop

do ... while loop

The do-while loop is a variation of the while loop.

Its syntax is:

```
do {
    // loop body;
    statement(s);
} while (loop-continuation-condition);
```
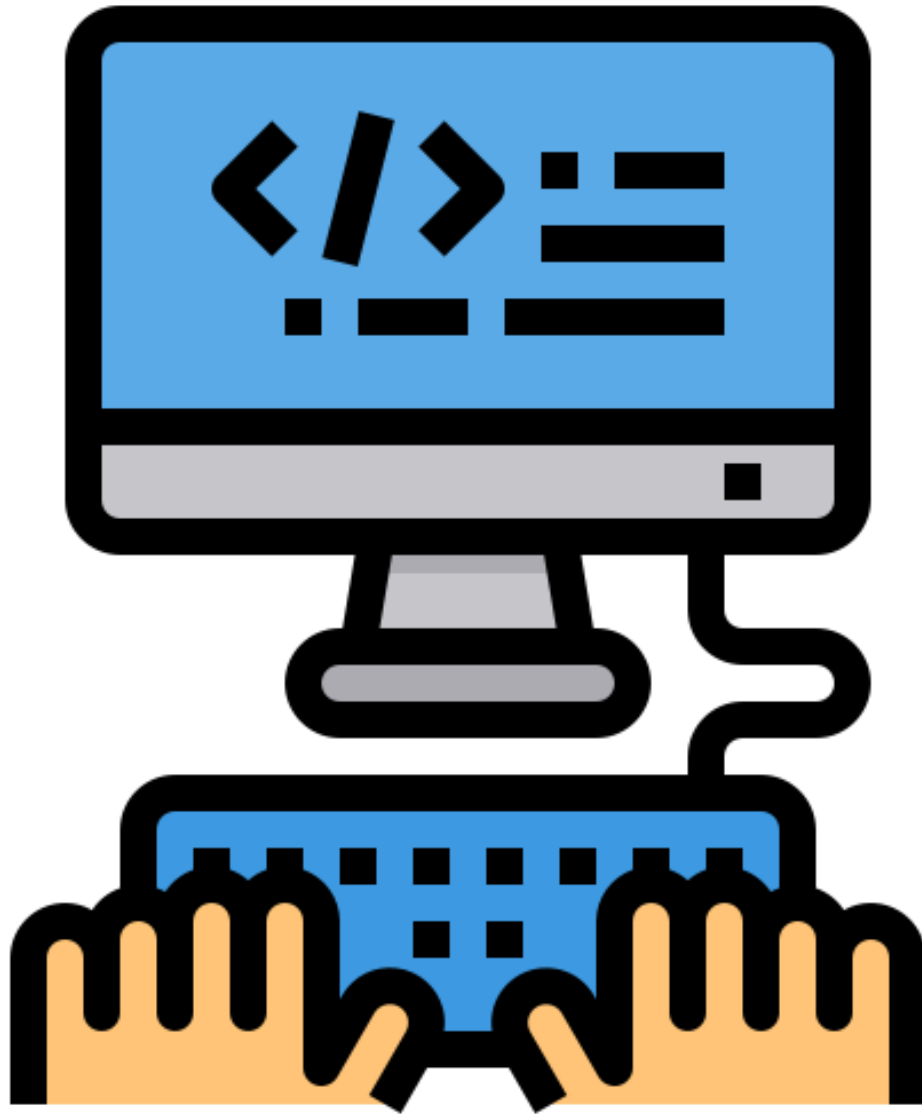

Flow Diagram of do .. while LOOP
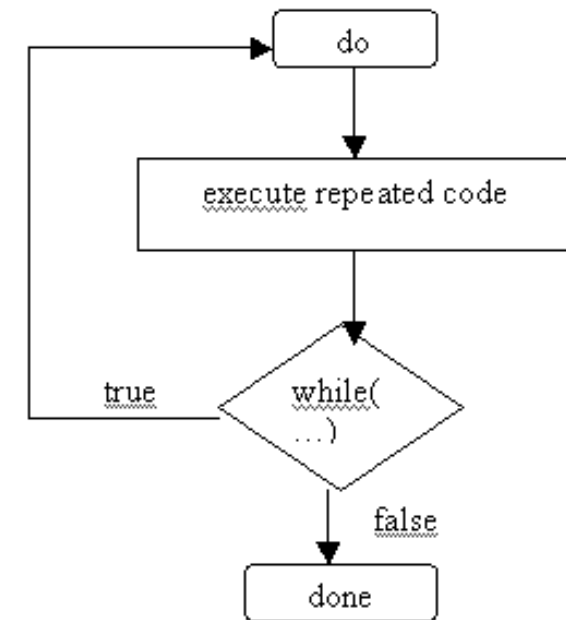
Difference between do-while-loop and while-loop

The difference between a while loop and a do-while-loop is the order in which the loop-continuation-condition is evaluated and the loop body executed. You can write a loop using either the while-loop or the do-while loop. Sometimes one is a more convenient choice than the other.

Number System

Section 7

Binary Value	Decimal Representation				Decimal Value
	8	4	2	1	
0 0 0 0	0 +	0 +	0 +	0	0
0 0 0 1	0 +	0 +	0 +	1	1
0 0 1 0	0 +	0 +	2 +	0	2
0 0 1 1	0 +	0 +	2 +	1	3
0 1 0 0	0 +	4 +	0 +	0	4
0 1 0 1	0 +	4 +	0 +	1	5
0 1 1 0	0 +	4 +	2 +	0	6
0 1 1 1	0 +	4 +	2 +	1	7
1 0 0 0	8 +	0 +	0 +	0	8
1 0 0 1	8 +	0 +	0 +	1	9
1 0 1 0	8 +	0 +	2 +	0	10

Binary to Decimal

Decimal, Binary, Octal, Hexidecimal Values

Decimal	Binary	Octal	Hexidecimal
0	0000	0	0
1	0001	1	1
2	0010	2	2
3	0011	3	3
4	0100	4	4
5	0101	5	5
6	0110	6	6
7	0111	7	7
8	1000	10	8
9	1001	11	9
10	1010	12	A
11	1011	13	B
12	1100	14	C
13	1101	15	D
14	1110	16	E
15	1111	17	F

Binary/Decimal Conversion
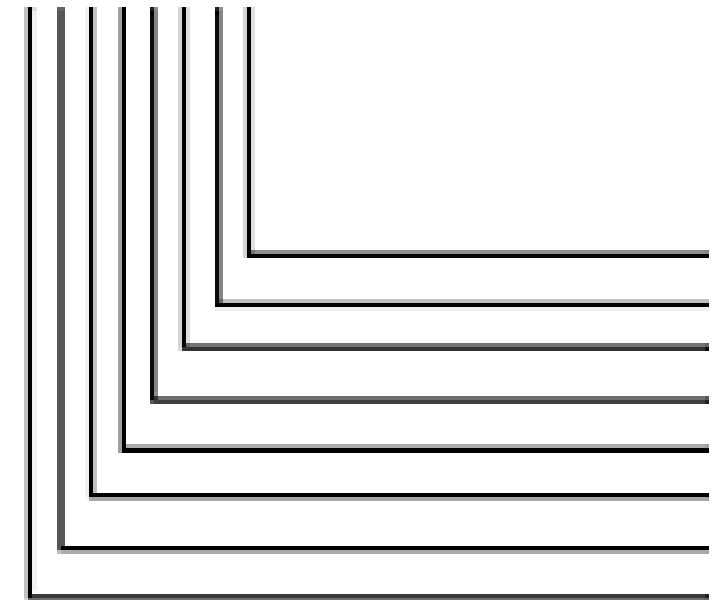
Decimal to Binary

$$156_{10} = 10011100_2$$

Divider	Dividend	Remainder
2	202	0
2	101	1
2	50	0
2	25	1
2	12	0
2	6	0
2	3	1
		1

Errors and Exceptions

Section 8
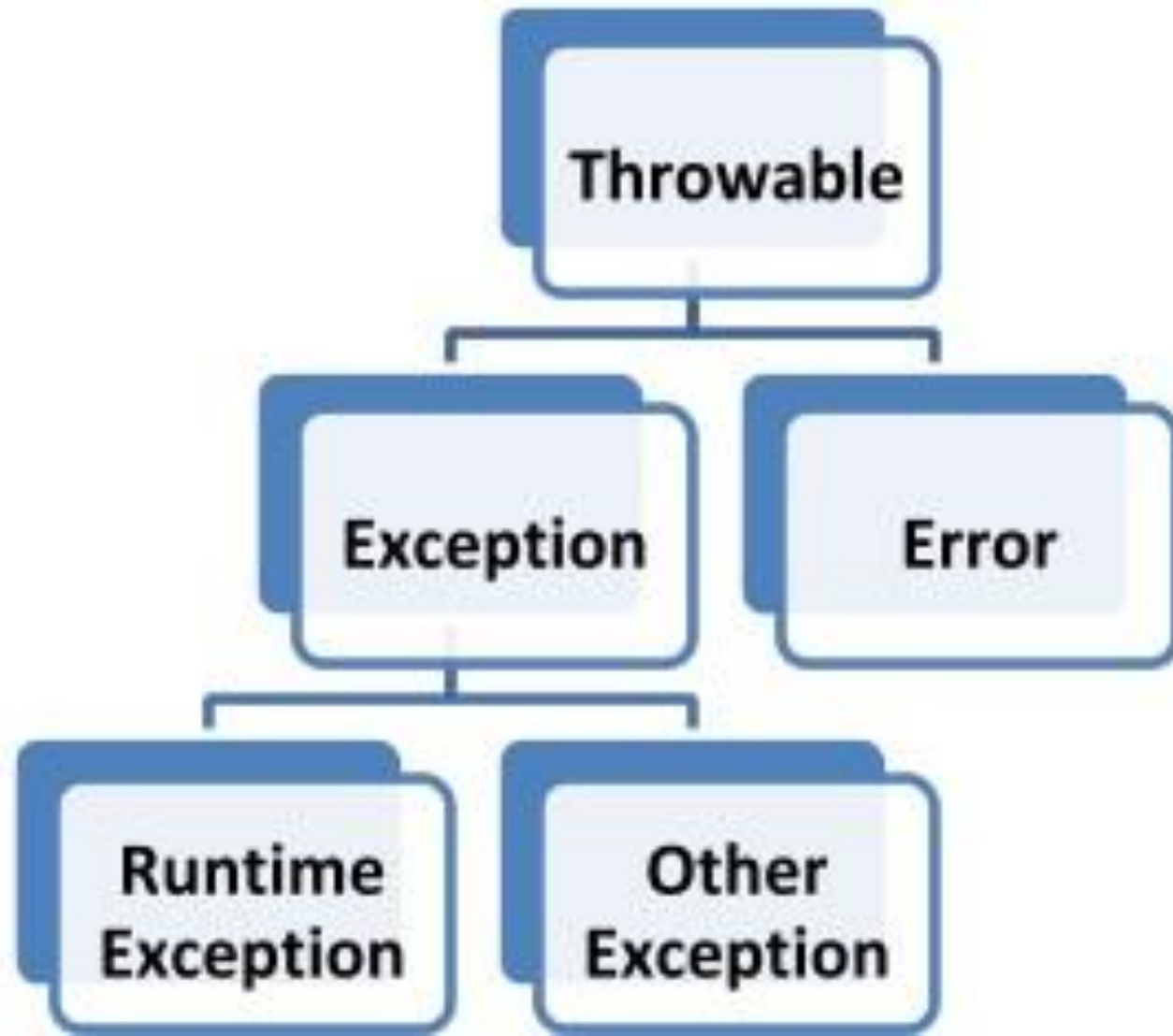
```java
try {
    int budget = 1000;
    System.out.println("Success");
}
catch(Exception ex) {
    System.out.println(ex);
}
finally {
    System.out.println("This always runs");
}
```

Errors and Exceptions

- An exception is an error condition that occurs during the execution of a Java program. For example, if you divide an integer by zero, an **ArithmeticException** will be thrown. If you use a negative array index, an **ArrayIndexOutOfBoundsException** will be thrown.

- An unchecked exception is one that is automatically handled by Java's standard exception handling methods, which terminate execution. It is thrown if an attempt is made to divide an integer by 0, or if an array index goes out of bounds, and so on. The exception tells you that you now need to fix your code!

Errors and Exceptions

- A checked exception is one where you provide code to handle the exception, either a try/catch/finally statement, or an explicit throw new ... Exception clause. These exceptions are not necessarily caused by an error in the code. For example, an unexpected end-of-file could be due to a broken network connection.

- Checked exceptions are not part of the AP Java subset.

Exception
ArithmeticException
NullPointerException
ArrayIndexOutOfBoundsException
IndexOutOfBoundsException
IllegalArgumentException
ConcurrentModificationException

The following unchecked exceptions are in the AP Java subset:

Example 1

```
if (nurnScores 0)
    throw new ArithmeticException("Cannot divide by zero");

else
    findAverageScore();
```

Example 2

```
public void setRadius(int newRadius) {
    if (newRadius < 0)
        throw new IllegalArgumentException ("Radius cannot be negative");
    else
        radius = newRadius;
}
```