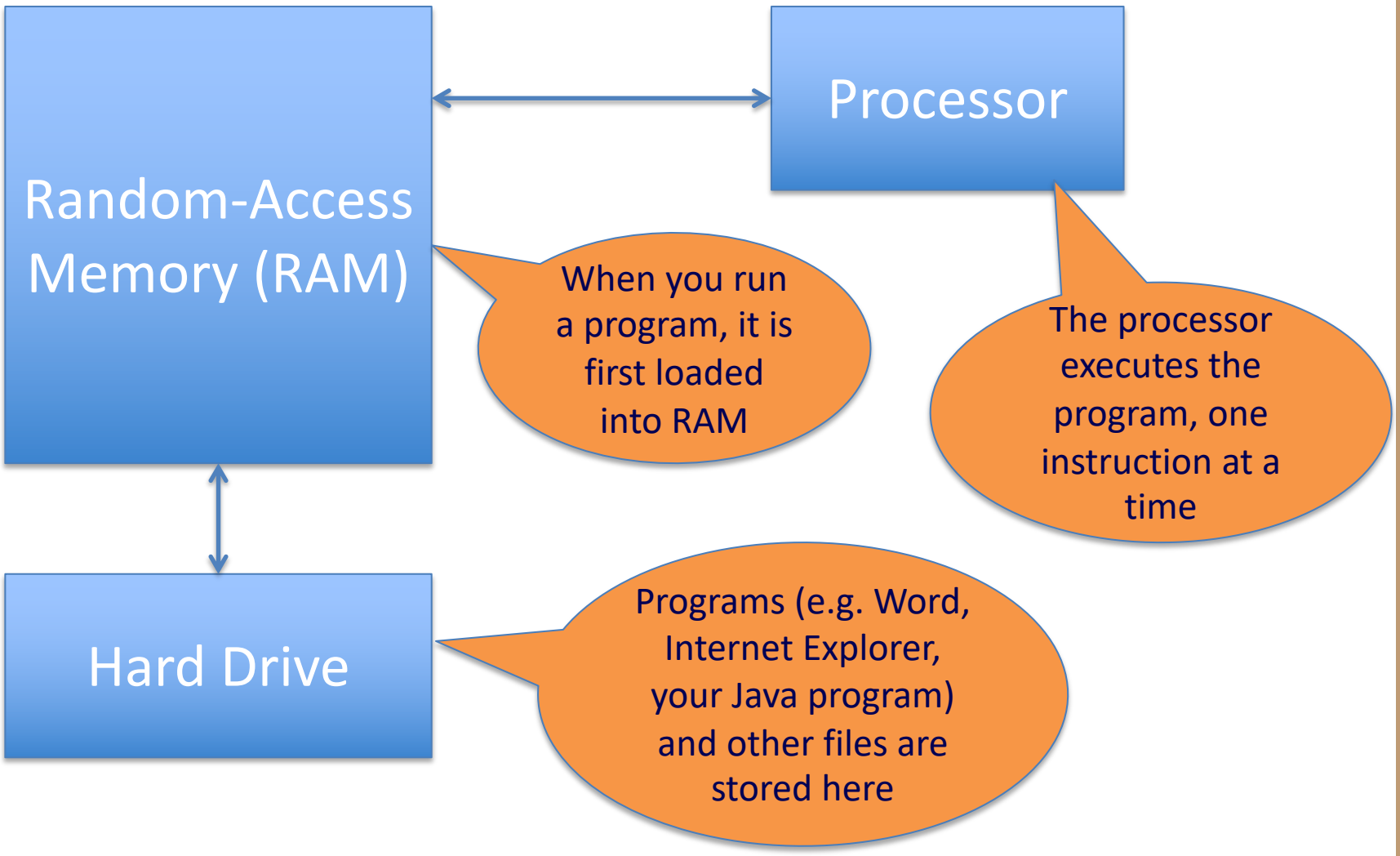
Getting Started with Java

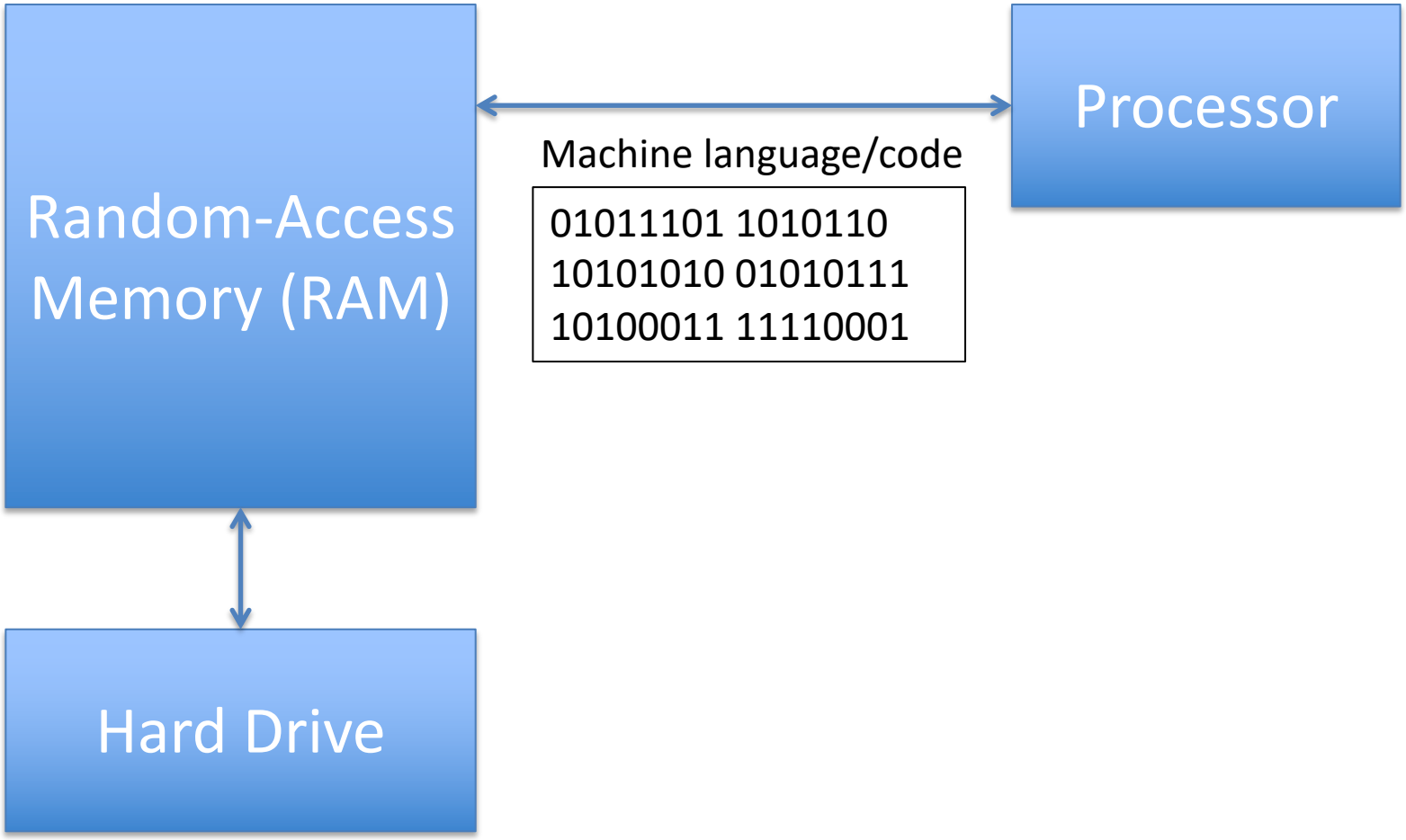
ASHESI

Organization of a Computer



ASHESI

What Language Does the Computer Processor Understand?



ASHESI

But … we don't write machine language, so how does this work?

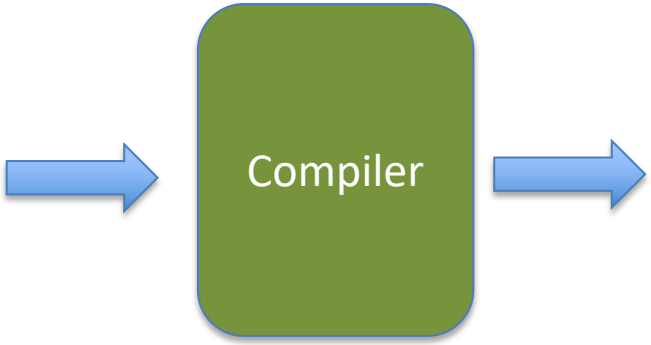
```java
public class Sum1
{
  public static void main(String[] args)
  {
    System.out.println("I am going to add 5 and 3");

    // declare variables to hold the numbers
    int n1, n2;

    // assign the numbers
    n1 = 5;
    n2 = 3;

    // print out the sum
    System.out.print("The sum of 5 and 3 is: ");
```

Our program, written in a high-level language



01011101 1010110
10101010 01010111
10100011 11110001

Machine code for our specific type of processor

Actually, this is a slightly simplified. The translation actually happens over several steps, e.g.: High level language ➔ Assembly language ➔ Machine code.

The "compiler" here actually consists of a compiler, an assembler and a linker. But you'll learn more about that in a different class!

ASHESI

Some Programs are interpreted, not compiled

```
public class Sum1
{
  public static void main(String[] args)
  {
    System.out.println("I am going to add 5 and 3");

    // declare variables to hold the numbers
    int n1, n2;

    // assign the numbers
    n1 = 5;
    n2 = 3;

    // print out the sum
    System.out.print("The sum of 5 and 3 is: ");
```

Our program, written in a high-level language

One instruction at a time

Interpreter

01011101 1010110

Single instruction at a time in machine code for our specific type of processor

ASHESI

Java is a bit of a special case

```java
public class Sum1
{
  public static void main(String[] args)
  {
    System.out.println("I am going to add 5 and 3");

    // declare variables to hold the numbers
    int n1, n2;

    // assign the numbers
    n1 = 5;
    n2 = 3;

    // print out the sum
    System.out.print("The sum of 5 and 3 is: ");
```

Our Java program

Java Compiler

Java Bytecode -- Machine code for a "virtual machine"

ADDAB LOADX SAVEY MOVEX MOVEB ADDXY

One instruction at a time

Java Bytecode Interpreter (Java Virtual Machine – JVM)

01011101 1010110

Single instruction at a time in machine code for our specific type of processor

ASHESI

Compiling & Running Java Programs

- Many IDEs support Java Development
- You can use an IDE of your choice. If you don't yet have one, we suggest VSCode
- You can also compile programs at the command-line using the **javac** command and then run them using the **java** command
  - Example: **javac** MyProgram.java
  - **java** MyProgram

ASHESI

Comparing Syntax: Python versus Java

```python
num1 = 3
num2 = 4

ans = num1 + num2

print("Answer is:")
print(ans)
```

Python: sum1.py

```java
public class Sum1 {
    public static void main(String[] args) {
        int num1;
        int num2;
        int ans;

        num1 = 3;
        num2 = 4;
        ans = num1 + num2;

        System.out.println("Answer is:");
        System.out.println(ans)
    }
}
```

Java: Sum1.java

ASHESI

Comparing Syntax: Python versus Java

	Variable declarations	int num1; int num2; int ans;
num1 = 3 num2 = 4	Variable assignments	num1 = 3; num2 = 4;
ans = num1 + num2	Expressions & variable assignments	ans = num1 + num2;
print(“Answer is:”) print(ans)	Output	System.out.println(“Answer is:”); System.out.println(ans)

ASHESI

Basic Computation in Java

ASHESI

You try!

- Can you write a simple Java program that asks the user to enter their year of birth, and then it tells them how old they are?
- You can also tell them if they are an adult

ASHESI

Data Types in Java

- Two categories of data types in Java
  - Primitive types, e.g. int store some data

```
int num = 5 + 6;
int anotherNum = num – 3;
```

  - Class types, or Objects, e.g. Scanner have behaviours (methods) associated with them

```
Scanner keyboard = new Scanner(System.in);
 int num = keyboard.nextDouble();
```

ASHESI

Primitive Data Types - Integers

Type Name	Kind of Value	Memory Used	Number of Possible Values	Range of Values
byte	Integer	1 byte (8 bits)	2^8 (256)	-2^7 to (2^7-1) [-128 to 127]
short	Integer	2 bytes (16 bits)	2^{16} (65,536)	-2^{15} to $(2^{15}-1)$ [-32,768 to 32,767]
int	Integer	4 bytes (32 bits)	2^{32} (4,294,967,296)	-2^{31} to $(2^{31}-1)$ [-2,147,483,648 to 2,147,483,647]
long	Integer	8 bytes (64 bits)	2^{64} ($1.844674407370955 \times 10^{19}$)	-2^{63} to $(2^{63}-1)$

ASHESI

Primitive Data Types – Others

Type Name	Kind of Value	Memory Used	Number of Possible Values	Range of Values
float	Floating-point	4 bytes (32 bits)	2^{32} (4,294,967,296)	$\pm 3.40282347 \times 10^{38}$ to $\pm 1.40239846 \times 10^{-45}$
double	Floating-point	8 bytes (64 bits)	2^{64} ($1.844674407370955 \times 10^{19}$)	$\pm 3.40282347 \times 10^{308}$ to $\pm 1.40239846 \times 10^{-45}$
char	Single character (Unicode)	2 bytes (16 bits)	2^{16} (65,536)	Unicode values from 0 to (2^{16}-1) [0 to 65,535]
boolean	True or false	1 bit	2	true or false

ASHESI

Binary Operators

- Binary operators:
  - Addition +, Subtraction -, Multiplication *
  - Division /
    - Floating point division example: 3.0/2.0 = 1.5
    - Integer division example: 3/2 = 1
  - Remainder %
    - Example: 12 % 5 = 2

ASHESI

A Note on Testing to Find Errors

- Let's look at 3 types of errors we might have in our program:
  1. Compiler / syntax errors
     - E.g. having an extra character after the semi-colon
     - The compiler will catch these errors and complain
  2. Run-time errors
     - E.g. errors due to bad user input (e.g. dividing a number by zero, given a fractional number when an int is required)
     - During runtime (i.e. when you're running your program), Java will "throw an exception" (give an error)
  3. Logical errors
     - E.g. using integer division when I really want floating point (double) division
     - There might not be an obvious indication of these types of errors, other than getting the wrong answer

ASHESI

Some more operators on integers

- Specialized assignment operators, just like in Python:

  `myNum += y` is shorthand for `myNum = myNum + y`

  Similarly, we can say

  `myNum -= y` instead of `myNum = myNum - y`

  `myNum *= y` instead of `myNum = myNum * y`

  `myNum /= y` instead of `myNum = myNum / y`

  `myNum %= y` instead of `myNum = myNum % y`

ASHESI

Unary Operators

- Increment & decrement operators:

  `myNum++` is shorthand for `myNum = myNum + 1`

  Similarly, we can say
  `myNum--` instead of `myNum = myNum - 1`

ASHESI

See example code: Widening.java

Variable Widening

- You can legally assign a value of a given type to a variable of a "wider" type
- The value is automatically widened

```
double price = 3;


short number1 = -234;
int number2 = number1;


char letterC= 'C';
int letterCUnicode = letterC;
```

Assign int to double. *price* now stores 3.0

Assign short to int

Assign char to int. *letterCUnicode* now stores the Unicode value of the character 'C'

ASHESI

See example code: Typecasting.java

Type Casting

- Using type casting, you can also explicitly change a value from a wider type and store it in a narrower type
  - Note that some information may be lost!

```
double price = 3.47;

int priceAsInt = (int) price;
```

Cast double to an int. *priceAsInt* now stores 3

ASHESI

More on Type Casting

- You can also explicitly cast a value as a different type for other reasons. E.g. if you're doing some arithmetic on two integer variables, but you want the answer to be a double, you can cast one or both of the numbers to a double

```
int numStudents = 55, numTeachers = 3;
double teacherStudentRatio;

teacherStudentRatio = numStudents/numTeachers;
```

Without typecasting, this expression is evaluated with integer division so teacherStudentRatio = 18

```
int numStudents = 55, numTeachers = 3;
double teacherStudentRatio;

teacherStudentRatio = numStudents/(double)numTeachers;
```

With typecasting, floating point division is used so teacherStudentRatio = 18.33

ASHESI

Exploring Characters

- Remember every character has a Unicode value
- When you cast a char to an int, you get its Unicode value
- You can use the operators '+', '-' and '==' on characters. What do you think will be the result?

- See ExploringCharacters.java

ASHESI

Strings

- String literals:
  - `“Hi, how are you?”`
  - `“I told you ‘hi’, but you didn’t hear me”`
- String variables:
  - `String greeting = “hi”`
- Concatenation of strings:
  - `String secondGreeting = greeting + “ there”.`
- Escape characters:
  - `String question = “Can you say \”hi\” to me?”`

ASHESI

Strings, continued

- Length of strings

```
int len = secondGreeting.length()
```

- String indices

0	1	2	3	4	5	6	7
h	i		t	h	e	r	e

```
System.out.println("The character at index 4 is " +
                   secondGreeting.charAt(4))
```

- See StringFun.java, Story.java

ASHESI

Other String Functions

- equals() – tests if two Strings are equal (case-sensitive)
- equalsIgnoreCase() – tests if two Strings are equal (case-insensitive)
- toLowerCase() – converts to lowercase
- toUpperCase() – converts to uppercase

- See StringFun2.java
- Look up more functions – read the Java API for Strings: http://download.oracle.com/javase/6/docs/api/index.html?java/lang/String.html

ASHESI

Other String Functions

- matches() – check if string matches a *regular expression*

- See StringFun3.java
- Learn about regular expressions and the Java Pattern class - http://download.oracle.com/javase/1.4.2/docs/api/index.html

ASHESI

Named Constants in Java

- Use for values that are not going to change in your program
- Convention is to name with all capital letters, with multiple words separated by underscores
- Defined inside the program, but outside the main() method

```
public static final int PESEWAS_PER_CEDI = 100;
public static final int SPEED_OF_SOUND = 331;
public static final double PI = 3.14;
```

- See CircleComputations.java

ASHESI

Commenting conventions

- // indicates that the rest of the current line is a comment

```
public class Score{

    public static void main(String[] args) {

        // the score provided by the user
        double originalScore;

        // the score after it has been adjusted by the instructor
        double adjustedScore;

        Scanner input = new Scanner(System.in);
        originalScore = input.nextDouble();

        adjustedScore = originalScore+5; // adjust the score
    }
}
```

ASHESI

Commenting conventions

- /* … */ indicates that the enclosed block is a comment

```
public class Score{

    public static void main(String[] args) {

        // the score provided by the user
        double originalScore;

        /* The score after it has been adjusted by the instructor.
        The instructor is allowed to adjust the score at his/her
        discretion */
        double adjustedScore;

        Scanner input = new Scanner(System.in);
        originalScore = input.nextDouble();

        adjustedScore = originalScore+5; // adjust the score
    }
}
```

ASHESI

Commenting Conventions

- /** … */ indicates a comment block to be processed by the Javadoc program
  - Before your class name, include a Javadoc comment briefly describing your program and specifying its author

```
/**
 * This is a program to compute how much change to give
 *
 * @author Ayorkor Korsah
 */
public class GivingChange {

 . . .

}
```

ASHESI

Other style guidelines - Indentation

- Indent each block of code demarcated by curly braces { } by **one** level relative to its parent block

```
public class Score{

    public static void main(String[] args) {
        // the provided by the user
        double originalScore;

        // the score after it has been adjusted by the instructor
        double adjustedScore;

        Scanner input = new Scanner(System.in);
        originalScore = input.nextDouble();

        if (originalScore < 10){
            adjustedScore = originalScore+5; // adjust the score
        }
    }
}
```

ASHESI

Other style guidelines - Indentation

- Example of **BAD** indentation because the code has poor readability; but it doesn't stop the programme from working

```
public class Score{

public static void main(String[] args) {
// the provided by the user
double originalScore;

// the score after it has been adjusted by the instructor
double adjustedScore;

Scanner input = new Scanner(System.in);
originalScore = input.nextDouble();

if (originalScore < 10){
adjustedScore = originalScore+5; // adjust the score
}
}
}
```

ASHESI

Other style guidelines - Indentation

- Example of **BAD** indentation because the code has poor readability; but it doesn't stop the programme from working

```
public class Score{

    public static void main(String[] args) {
// the provided by the user
double originalScore;

        // the score after it has been adjusted by the instructor
        double adjustedScore;

            Scanner input = new Scanner(System.in);
    originalScore = input.nextDouble();

        if (originalScore < 10){
    adjustedScore = originalScore+5; // adjust the score
            }
    }
}
```

ASHESI

More style guidelines

- Use meaningful variable names

✓ `double area, circumference;`

✗ `double myVar1, myVar2;`

- Use named constants rather than literals wherever possible

…

✓
```
double radius, circumference;
radius = input.nextInt();
circumference = PI * radius;
```

…

✗
```
double radius, circumference;
radius = input.nextInt();
circumference = 3.14* radius;
```

ASHESI

Braces

- Both of these ways of positioning braces are fine

```
public class Score
{
  public static void main(String[] args)
  {
    // the provided by the user
    double originalScore;
    Scanner input = new Scanner(System.in);
    originalScore = input.nextDouble();
  }
}
```

```
public class Score {
  public static void main(String[] args) {
    // the provided by the user
    double originalScore;
    Scanner input = new Scanner(System.in);
    originalScore = input.nextDouble();
  }
}
```

ASHESI