# Introduction to Android and Basics of Java

CS260-Android App Development

Paul Cao

# The players

Android – Open source mobile OS developed ny the Open Handset Alliance led by Google. Based on Linux 2.6 kernel

iOS – Apple's proprietary mobile OS, iPhone, iPod Touch, iPad. Derived from OS X, very UNIX like

Symbian – acquired by Nokia 2008

Windows Phone 7 – Microsoft – Kin, discontinued 6 weeks after initial launch

Blackberry OS – RIM (Research in Motion), proprietary OS

# Open handset Alliance

- http://www.openhandsetalliance.com
- Device manufacturers, chipmakers, software company
- And by Google

- Open source!

# Ubiquity

Many devices available

Www.android.com

- Wear
- Phone
- Tablet
- TV
- ...

# What is Android

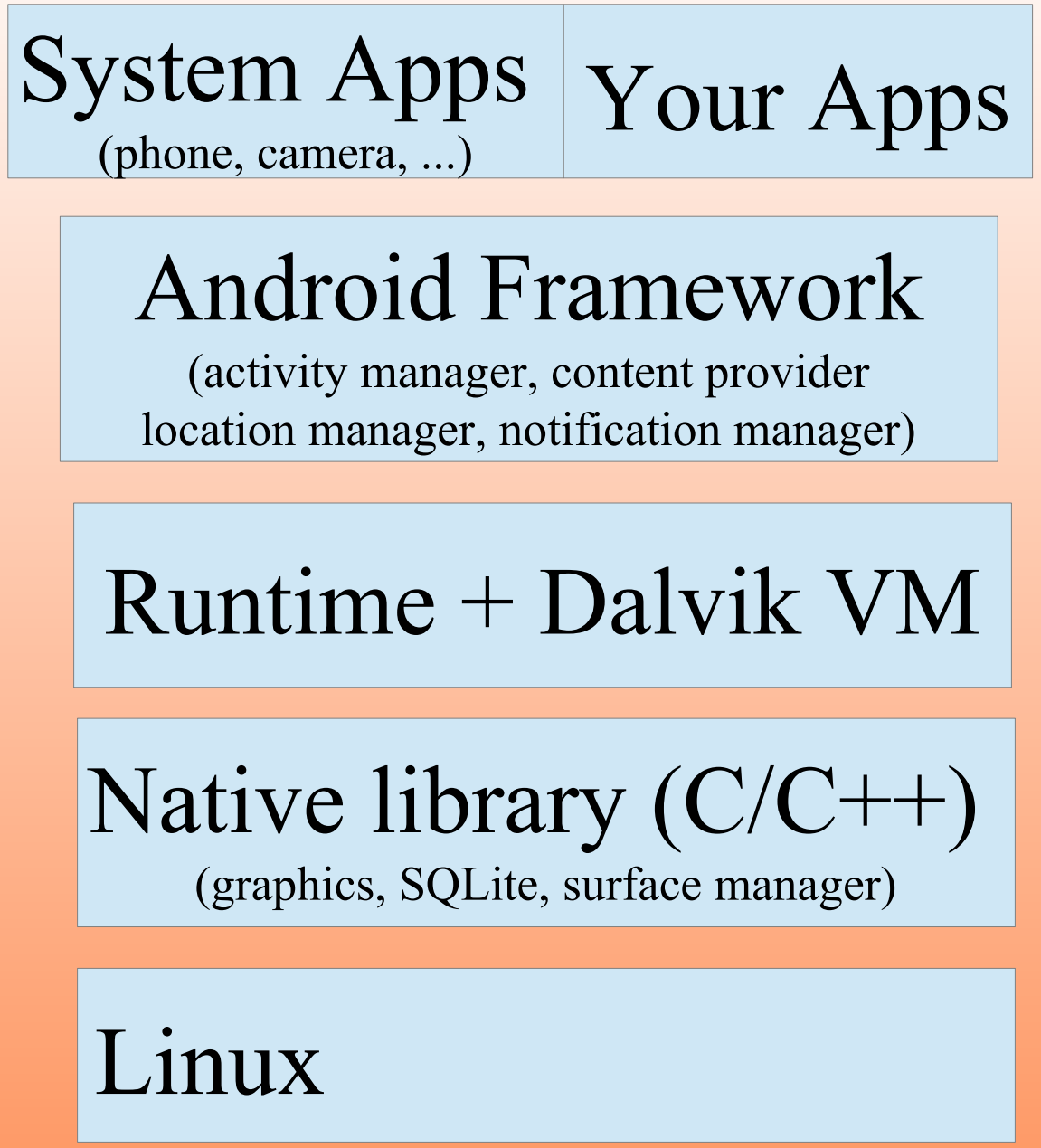
- Android is an open source operating system, created by Google specifically for use on mobile devices (cell phones and tablets)
- Linux based (2.6 kernel)
- Can be programmed in C/C++ but most app development is done in Java (Java access to C Libraries via JNI (Java Native Interface))
- Supports Bluetooth, Wi-Fi, and 4G networking

# Open source framework

Open it! http://source.android.com

SDK: http://developer.android.com

# Platform


System Apps
(phone, camera, ...)
Your Apps
Android Framework
(activity manager, content provider
location manager, notification manager)
Runtime + Dalvik VM
Native library (C/C++)
(graphics, SQLite, surface manager)
Linux

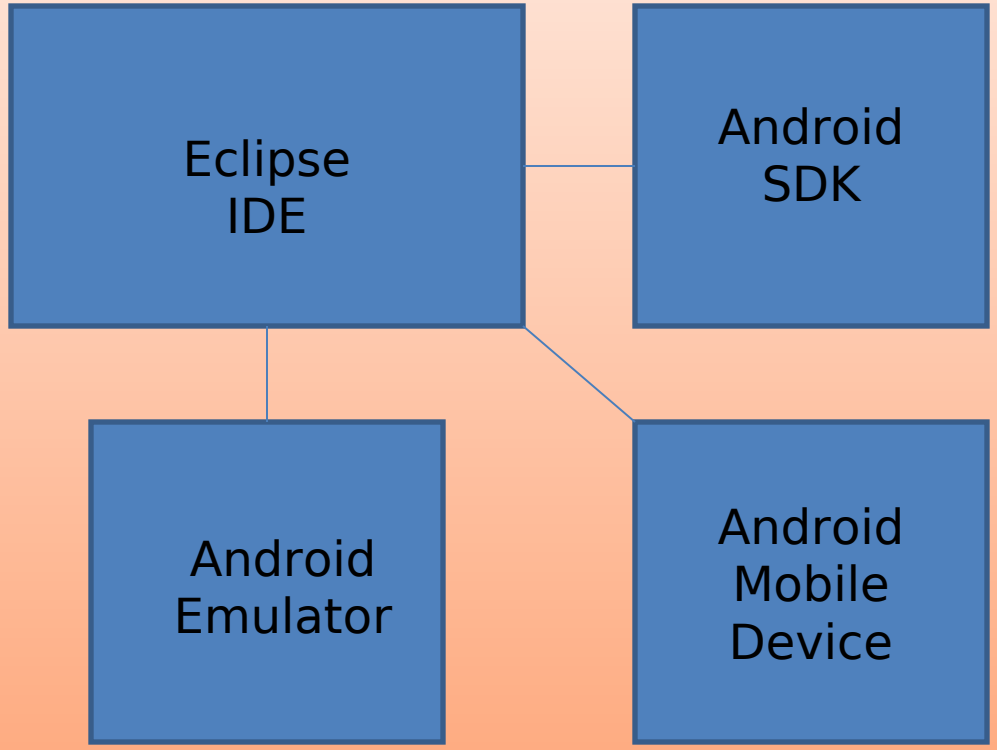
# Why develop for Mobile

- Smart phone and tablets are more popular and more powerful now
- Network (wifi and 4G/5G networks) are faster and faster
- Do you still use your phone just for calling someone?

# Why Android

- No certification required
- Google play provides “free” service for you to make money
- No approval process for application distribution
- Can provide powerful APIs (google map, background service, all apps are created equal)

# Android Application Development



# What you need to begin

- A relatively new computer (Windows/Mac/Linux)
- Eclipse with ADT
  - Java JDK (development kit, not JRE)
  - Android Development Kit (ADT)

# Plan for this course

Introduction to Java programming

- Flow control
- Methods
- Class and objects
- Collections
- Exception handling

Android App development

- Widgets
- Layout
- Preference
- Location
- Sensor
- Camera
- Data and file operation

# Get your hands wet

Hello World App

# First Java Program

- There are some concepts that are common to virtually all programming languages.
- Common concepts:
  - Key words
  - Operators
  - Punctuation
  - Programmer-defined identifiers
  - Strict syntactic rules.

# Programming Languages

## Sample Program

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

# Programming Languages
## Sample Program

- Key words in the sample program are:
  - `public`
  - `class`
  - `static`
  - `void`
- Key words are lower case (Java is a case sensitive language).
- Key words cannot be used as a programmer-defined identifier.

# Programming Languages

- Semi-colons are used to end Java statements; however, not all lines of a Java program end a statement.

# Programming Languages
## Variables

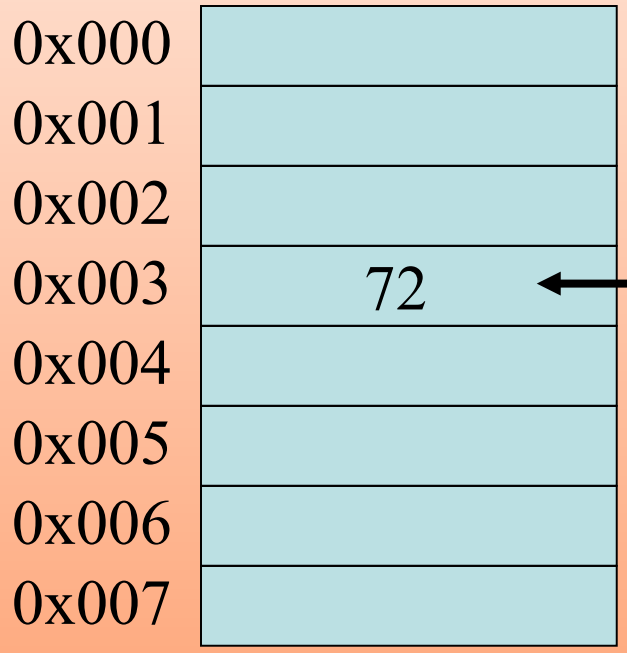
- Data in a Java program is stored in memory.
- Variable names represent a location in memory.
- Variables in Java are sometimes called fields.
- Variables are created by the programmer who assigns it a programmer-defined identifier.

**example:** `int hours = 40;`

- In this example, the variable *hours* is created as an integer (more on this later) and assigned the value of 40.

# Programming Languages
## Variables

The Java Virtual Machine (JVM) actually decides where the value will be placed in memory.




**Assume that the this variable declaration has been made.**

```
int length = 72;
```

**The variable length is a symbolic name for the memory location 0x003.**

# The Compiler and the Java Virtual Machine

- A *text editor* is used to edit and save a Java *source code file.* → *eclipse*
- Source code files have a *.java* file extension.
- A *compiler* is a program that translates source code into an executable form.

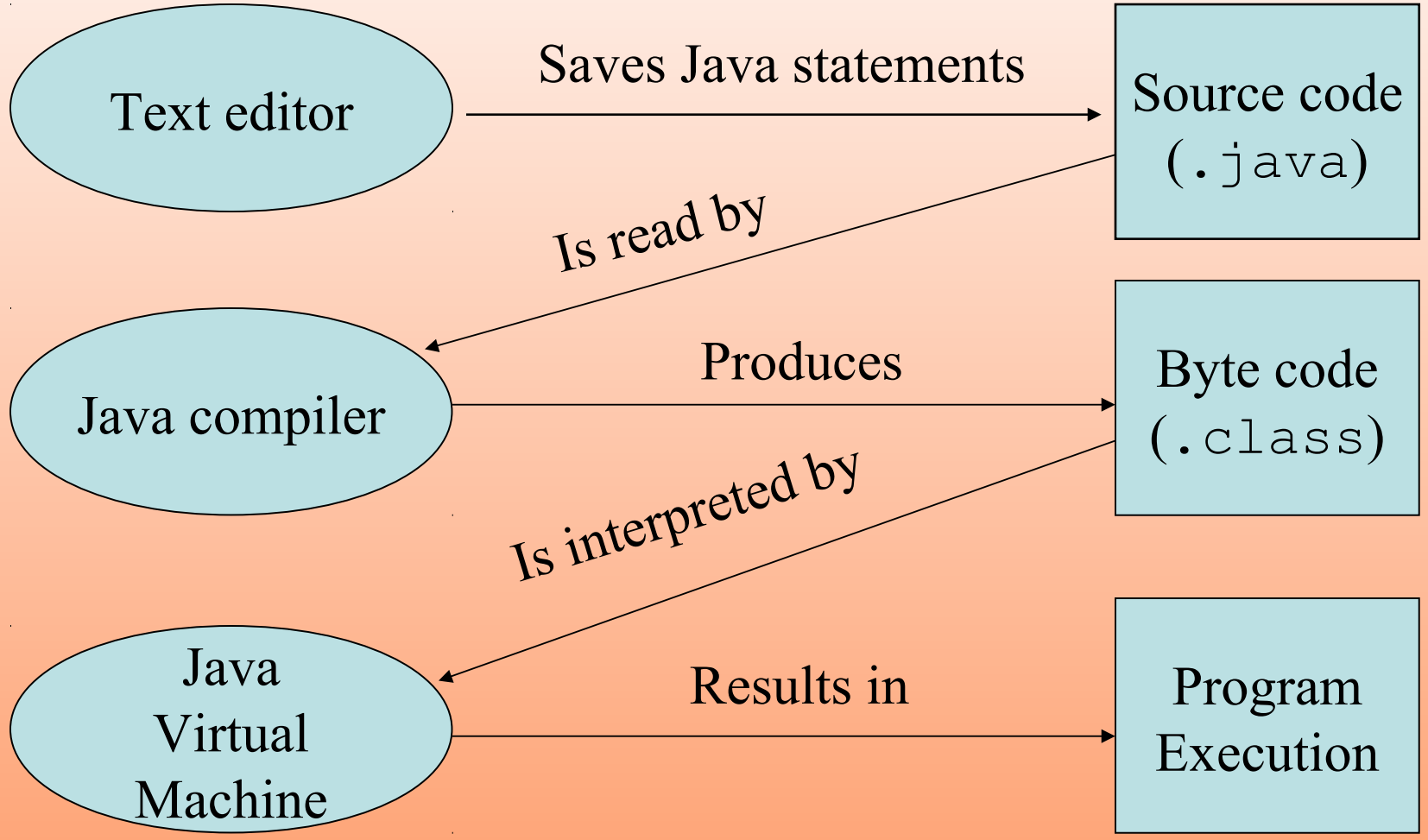
# The Compiler and the Java Virtual Machine

- Most compilers translate source code into *executable* files containing *machine code*.
- The Java compiler translates a Java source file into a file that contains *byte code* instructions.
- Byte code instructions are the machine language of the *Java Virtual Machine (JVM)* and cannot be directly executed directly by the CPU.

# The Compiler and the Java Virtual Machine

- Byte code files end with the *.class* file extension.
- The JVM is a program that *emulates* a micro-processor.
- The JVM executes instructions as they are read.
- JVM is often called an *interpreter*.
- Java is often referred to as an *interpreted language*.

# Program Development Process



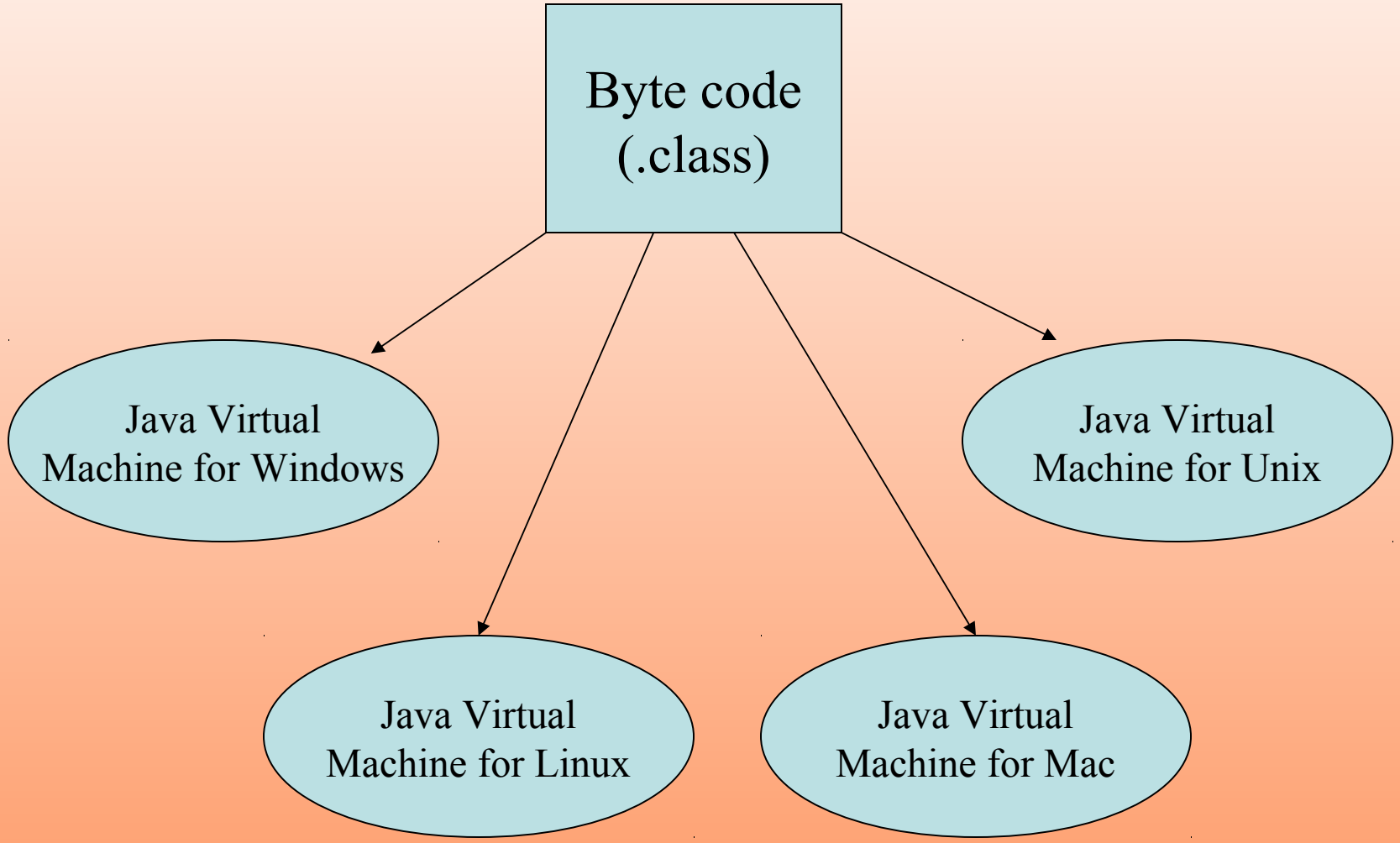
# Portability

- *Portable* means that a program may be written on one type of computer and then run on a wide variety of computers, with little or no modification.
- Java byte code runs on the JVM and not on any particular CPU; therefore, compiled Java programs are highly portable.
- JVMs exist on many platforms:
  - Windows
  - Mac
  - Linux
  - Unix
  - BSD
  - Etc.

# Portability

- With most programming languages, portability is achieved by compiling a program for each CPU it will run on.
- Java provides an JVM for each platform so that programmers do not have to recompile for different platforms.

# Portability



# Java Versions

- The software you use to write Java programs is called the Java Development Kit, or JDK.
- There are different editions of the JDK:
  - Java SE - Java2 *Standard Edition*.
  - Java EE - Java2 *Enterprise Edition*.
  - Java ME - Java2 *Micro Edition*.

Available for download at
http://www.oracle.com/technetwork/java

# Compiling a Java Program

- The Java compiler is a *command line* utility.
- The command to compile a program is:

  **javac filename.java**

- javac is the Java compiler.
- The `.java` file extension must be used.

Example: To compile a java source code file named Payroll.java you would use the command:

```
javac Payroll.java
```

# Hello world for CS260

Steps

1. Start Eclipse (this icon ). Select the default workspace (or pick your own dir)
2. Go to file menu → new → Java Project. Give it a name such as *project 2* or *my first program*. The click finish
3. Go to file menu → new → class. Give the class a name. **The file name and the class name must be the same!**
4. Type in your first Java code. Be careful with the syntax.
5. Click run or type ctrl+F11 to run your code. If there are errors, fix them and then run again.

# Parts of a Java Program

- A Java source code file contains one or more Java classes.
- If more than one class is in a source code file, only one of them may be public.
- The public class and the filename of the source code file must match.

  ex: A class named *Simple* must be in a file named *Simple.java*
- Each Java class can be separated into parts.

# Parts of a Java Program

See example: Simple.java

To compile the example:

- **`javac Simple.java`**
  - Notice the `.java` file extension is needed.
  - This will result in a file named *Simple.class* being created.

To run the example:

- **`java Simple`**
  - Notice there is no file extension here.
  - The *java* command assumes the extension is `.class`.

# Analyzing The Example

```
// This is a simple Java program.

public class Simple
{

}
```

**This is a Java comment. It is ignored by the compiler.**

**This is the class header for the class Simple**

**This area is the body of the class Simple. All of the data and methods for this class will be between these curly braces.**

# Analyzing The Example

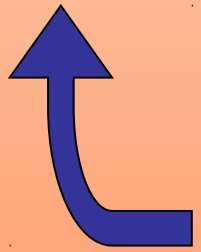
```
// This is a simple Java program.
public class Simple
{
    public static void main(String[] args)
    {

    }
}
```

**This is the method header for the main method. The main method is where a Java application begins.**

**This area is the body of the main method. All of the actions to be completed during the main method will be between these curly braces.**

# Analyzing The Example

```
// This is a simple Java program.
public class Simple
{
    public static void main(String [] args)
    {
        System.out.println("Programming is great fun!");
        System.out.println("Don't get bitten by bugs!");
    }
}
```



**This is the Java Statement that is executed when the program runs.**

# Parts of a Java Program

Comments

- The line is ignored by the compiler.
- The comment in the example is a single-line comment.

Class Header

- The class header tells the compiler things about the class such as what other classes can use it (public) and that it is a Java class (class), and the name of that class (Simple).

Curly Braces

- When associated with the class header, they define the scope of the class.
- When associated with a method, they define the scope of the method.

# Parts of a Java Program

- The `main` Method
  - This line must be exactly as shown in the example (except the *args* variable name can be programmer defined).
  - This is the line of code that the *java* command will run first.
  - This method starts the Java program.
  - Every Java *application* must have a `main` method.
- Java Statements
  - When the program runs, the statements within the `main` method will be executed.
  - Can you see what the line in the example will do?

# Java Statements

- If we look back at the previous example, we can see that there are only two line that ends with a semi-colon.

```
System.out.println("Programming is great fun!");
System.out.println("Don't get bitten by bugs!");
```

- This is because they are the only Java statements in the program.
- The rest of the code is either a comment or other Java framework code.

# Java Statements

Comments are ignored by the Java compiler so they need no semi-colons.

Other Java code elements that do not need semi colons include:

- class headers
  - Terminated by the code within its curly braces.
- method headers
  - Terminated by the code within its curly braces.
- curly braces
  - Part of framework code that needs no semi-colon termination.

# Short Review

Java is a case-sensitive language.

All Java programs must be stored in a file with a .java file extension.

Comments are ignored by the compiler.

A .java file may contain many classes but may only have one public class.

If a .java file has a public class, the class must have the same name as the file.

# Short Review

Java applications must have a `main` method.

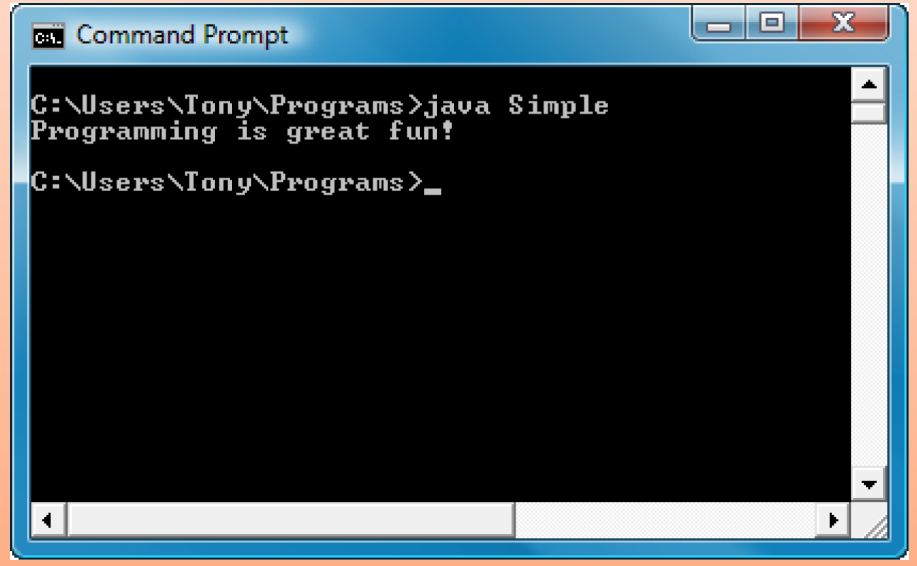
For every left brace, or opening brace, there must be a corresponding right brace, or closing brace.

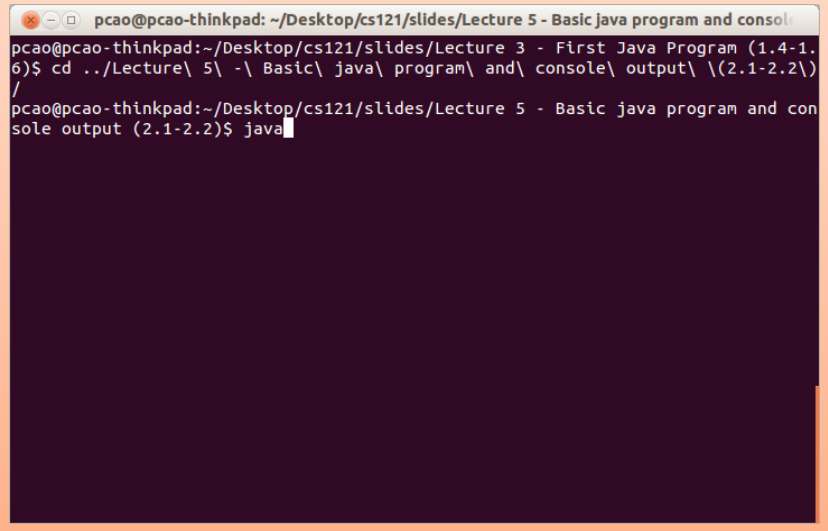
Statements are terminated with semicolons.

- Comments, class headers, method headers, and braces are not considered Java statements.

# Console Output

Many of the programs that you will write will run in a console window.



# Console Output

The console window that starts a Java application is typically known as the *standard output* device.

The *standard input* device is typically the keyboard.

Java sends information to the standard output device by using a Java class stored in the standard Java library.

# Console Output

Java classes in the standard Java library are accessed using the Java Applications Programming Interface (API).

The standard Java library is commonly referred to as the *Java API*.

# Console Output

The previous example uses the line:

```
System.out.println("Programming is great fun!");
```

This line uses the `System` class from the standard Java library.

The `System` class contains methods and objects that perform system level tasks.

The `out` object, a member of the `System` class, contains the methods `print` and `println`.

# Console Output

The `print` and `println` methods actually perform the task of sending characters to the output device.

The line:

```
System.out.println("Programming is great fun!");
```

is pronounced: System dot out dot println …

The value inside the parenthesis will be sent to the output device (in this case, a string).

# Console Output

The `println` method places a newline character at the end of whatever is being printed out.

The following lines:

```
System.out.println("This is being printed out");
System.out.println("on two separate lines.");
```

Would be printed out on separate lines since the first statement sends a newline command to the screen.

# Console Output

The `print` statement works very similarly to the `println` statement.

However, the `print` statement does not put a newline character at the end of the output.

The lines:

```
System.out.print("These lines will be");
System.out.print("printed on");
System.out.println("the same line.");
```

Will output:

```
These lines will beprinted onthe same line.
```

Notice the odd spacing? Why are some words run together?

# Console Output

For all of the previous examples, we have been printing out strings of characters.

There are some special characters that can be put into the output.

```
System.out.print("This line will have a newline at the end.\n");
```

The `\n` in the string is an escape sequence that represents the newline character.

Escape sequences allow the programmer to print characters that otherwise would be unprintable.

# Java Escape Sequences

| | | |
|---|---|---|
| `\n` | newline | Advances the cursor to the next line for subsequent printing |
| `\t` | tab | Causes the cursor to skip over to the next tab stop |
| `\b` | backspace | Causes the cursor to back up, or move left, one position |
| `\r` | carriage return | Causes the cursor to go to the beginning of the current line, not the next line |
| `\\` | backslash | Causes a backslash to be printed |
| `\'` | single quote | Causes a single quotation mark to be printed |
| `\"` | double quote | Causes a double quotation mark to be printed |

# Java Escape Sequences

Even though the escape sequences are comprised of two characters, they are treated by the compiler as a single character.

```
System.out.print("These are our top sellers:\n");
System.out.print("\tComputer games\n\tCoffee\n ");
System.out.println("\tAspirin");
```

Would result in the following output:

```
These are our top seller:
        Computer games
        Coffee
        Asprin
```

With these escape sequences, complex text output can be achieved.

# Exercise

Can you write a Java program that generates the following shape?

Hint: Use \t to get good alignment of the hellos

```
                hello
        hello           hello
hello                           hello
        hello           hello
                hello
```

# Variables and Literals

A variable is a named storage location in the computer's memory.

A literal is a value that is written into the code of a program.

Programmers determine the number and type of variables a program will need.

See example:Variable.java

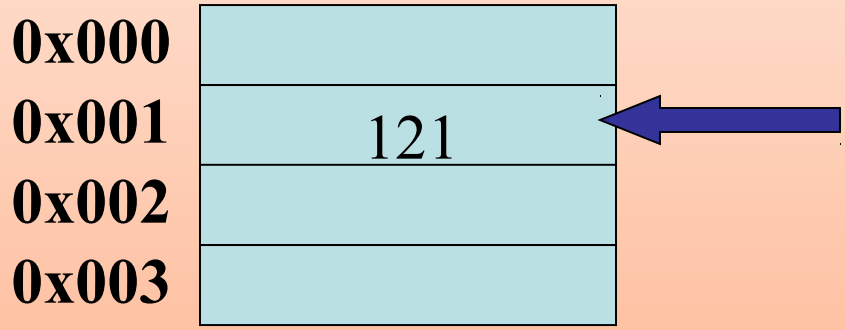
# Variables and Literals

This line is called a *variable declaration*.

```
int value;
```

The following line is known as an assignment statement.

```
value = 121;
```




The value 121 is stored in memory.

This is a string *literal*. It will be printed as is.

```
System.out.print("The value is ");
    System.out.println(value);
```

The integer 121 will be printed out here. Notice no quote marks?

# The + Operator

The + operator can be used in two ways.

- as a concatenation operator
- as an addition operator

If either side of the + operator is a string, the result will be a string.

```
System.out.println("Hello " + "World");
System.out.println("The value is: " + 5);
System.out.println("The value is: " + value);
System.out.println("The value is: " + '/n' + 5);
```

# String Concatenation

Java commands that have string literals must be treated with care.

A string literal value cannot span lines in a Java source code file.

```
System.out.println("This line is too long and now it
has spanned more than one line, which will cause a
syntax error to be generated by the compiler. ");
```

# String Concatenation

The String concatenation operator can be used to fix this problem.

```
System.out.println("These lines are " +
                   "are now ok and will not " +
                   "cause the error as before.");
```

String concatenation can join various data types.

```
System.out.println("We can join a string to " +
                   "a number like this: " + 5);
```

# String Concatenation

The Concatenation operator can be used to format complex String objects.

```
System.out.println("The following will be printed " +
                   "in a tabbed format: " +
                   \n\tFirst = " + 5 * 6 + ", " +
                   "\n\tSecond = "+(6 + 4) + "," +
                   "\n\tThird = " + 16.7 + ".");
```

Notice that if an addition operation is also needed, it must be put in parenthesis.

# Identifiers

Identifiers must follow certain rules:

- An identifier may only contain:
  - letters a–z or A–Z,
  - the digits 0–9,
  - underscores (_), or
  - the dollar sign ($)
- The first character may not be a digit.
- Identifiers are case sensitive.
  - `itemsOrdered` is not the same as `itemsordered`.
- Identifiers cannot include spaces.

# Java Reserved Keywords

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

# Variable Names

Variable names should be descriptive.

Descriptive names allow the code to be more readable; therefore, the code is more maintainable.

Which of the following is more descriptive?

```
double tr = 0.0725;
double salesTaxRate = 0.0725;
```

Java programs should be *self-documenting*.

# Java Naming Conventions

Variable names should begin with a lower case letter and then switch to title case thereafter:

Ex: `int caTaxRate`

Class names should be all title case.

Ex: `public class BigLittle`

More Java naming conventions can be found at:

http://java.sun.com/docs/codeconv/html/CodeConventions.doc8.html

A general rule of thumb about naming variables and classes are that, with some exceptions, their names tend to be nouns or noun phrases.

# Primitive Data Types

- Primitive data types are built into the Java language and are not derived from classes.
  - There are 8 Java primitive data types.
    - `byte`
    - `short`
    - `int`
    - `long`
    - `float`
    - `double`
    - `boolean`
    - `char`

# Numeric Data Types

| | | |
|---|---|---|
| byte | 1 byte | Integers in the range -128 to +127 |
| short | 2 bytes | Integers in the range of -32,768 to +32,767 |
| int | 4 bytes | Integers in the range of -2,147,483,648 to +2,147,483,647 |
| long | 8 bytes | Integers in the range of -9,223,372,036,854,775,808 to +9,223,372,036,854,775,807 |
| float | 4 bytes | Floating-point numbers in the range of ±3.410-38 to ±3.41038, with 7 digits of accuracy |
| double | 8 bytes | Floating-point numbers in the range of ±1.710-308 to ±1.710308, with 15 digits of accuracy |

# Floating Point Data Types

Data types that allow fractional values are called *floating-point* numbers.

- 1.7 and -45.316 are floating-point numbers.

In Java there are two data types that can represent floating-point numbers.

- `float` - also called *single precision* (7 decimal points).
- `double` - also called *double precision* (15 decimal points).

# Floating Point Literals

When floating point numbers are embedded into Java source code they are called *floating point literals*.

The default type for floating point literals is `double`.

- 29.75, 1.76, and 31.51 are `double` data types.

Java is a *strongly-typed* language.

See example: Sale.java

# Floating Point Literals

A `double` value is not compatible with a `float` variable because of its size and precision.

- `float number;`
- `number = 23.5; // Error!`

A `double` can be forced into a `float` by appending the letter F or f to the literal.

- `float number;`
- `number = 23.5F; // This will work.`

# Floating Point Literals

Literals cannot contain embedded currency symbols or commas.

- `grossPay = $1,257.00; // ERROR!`
- `grossPay = 1257.00;  // Correct.`

Floating-point literals can be represented in *scientific notation*.

- $47{,}281.97 == 4.728197 \times 10^4$.

Java uses *E notation* to represent values in scientific notation.

- $4.728197X10^4 == 4.728197E4$.

# Scientific and E Notation

| Decimal Notation | Scientific Notation | E Notation |
|---|---|---|
| 247.91 | $2.4791 \times 10^{2}$ | 2.4791E2 |
| 0.00072 | $7.2 \times 10^{-4}$ | 7.2E-4 |
| 2,900,000 | $2.9 \times 10^{6}$ | 2.9E6 |

See example: SunFacts.java

# The `boolean` Data Type

The Java `boolean` data type can have two possible values.

- `true`
- `false`

The value of a `boolean` variable may only be copied into a `boolean` variable.

# Unicode

Internally, characters are stored as numbers.

Character data in Java is stored as Unicode characters.

The Unicode character set can consist of 65536 ($2^{16}$) individual characters.

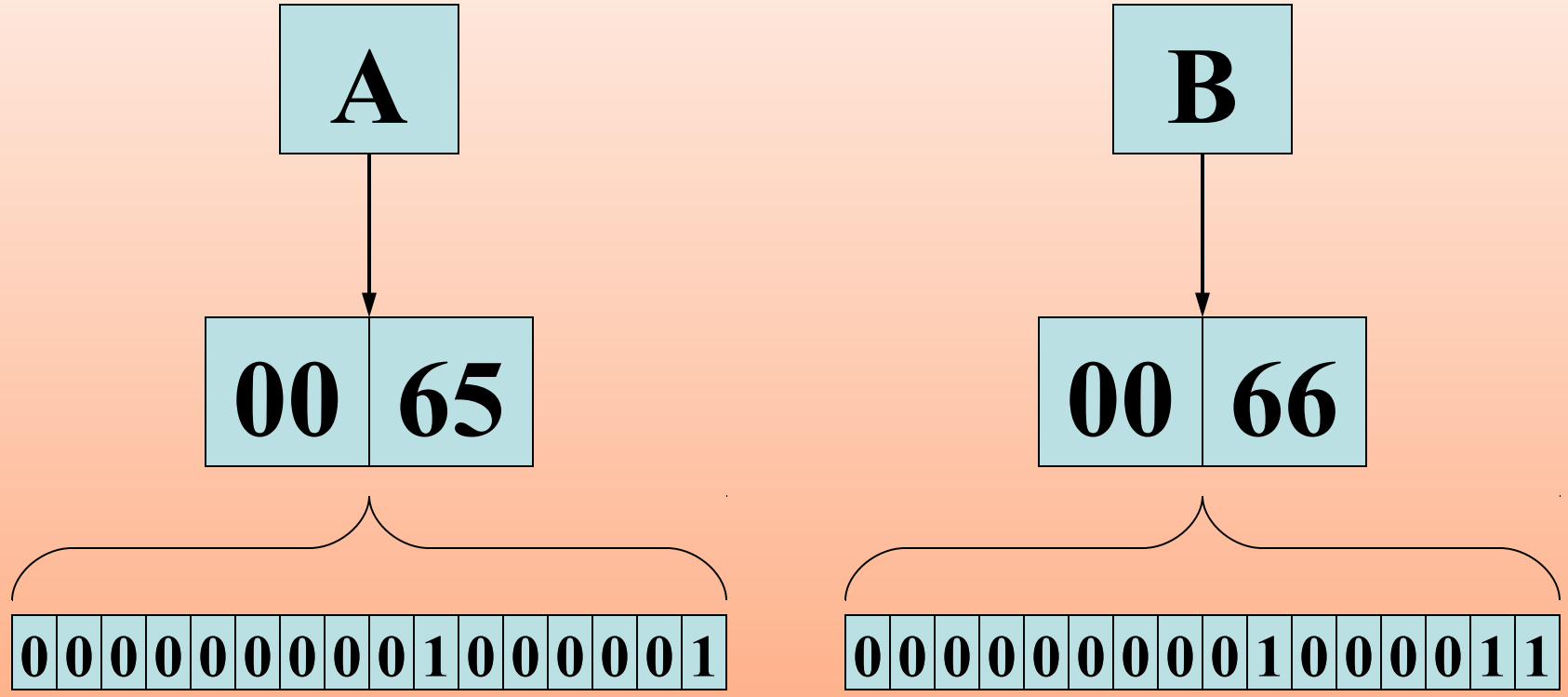
This means that each character takes up 2 bytes in memory.

The first 256 characters in the Unicode character set are compatible with the ASCII* character set.

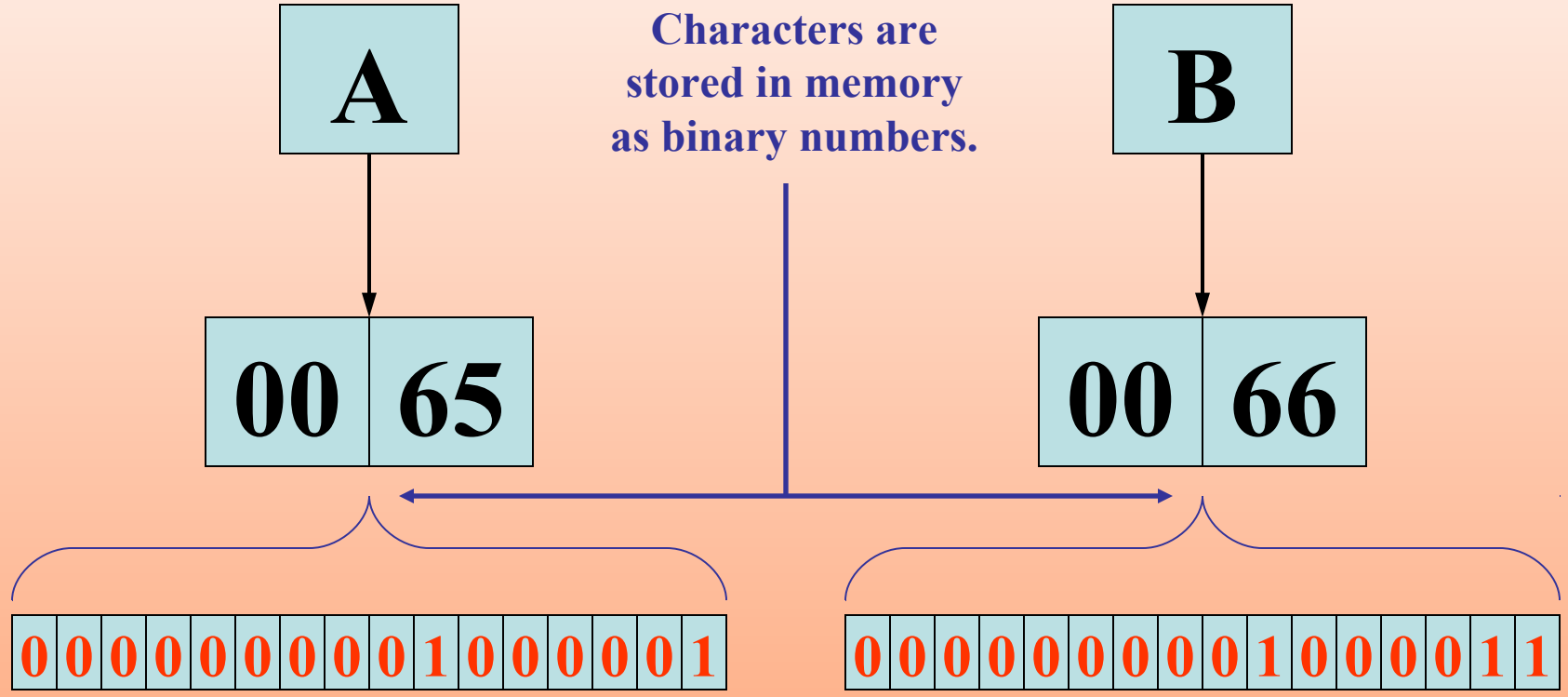
See example: Letters2.java

*American Standard Code for Information Interchange

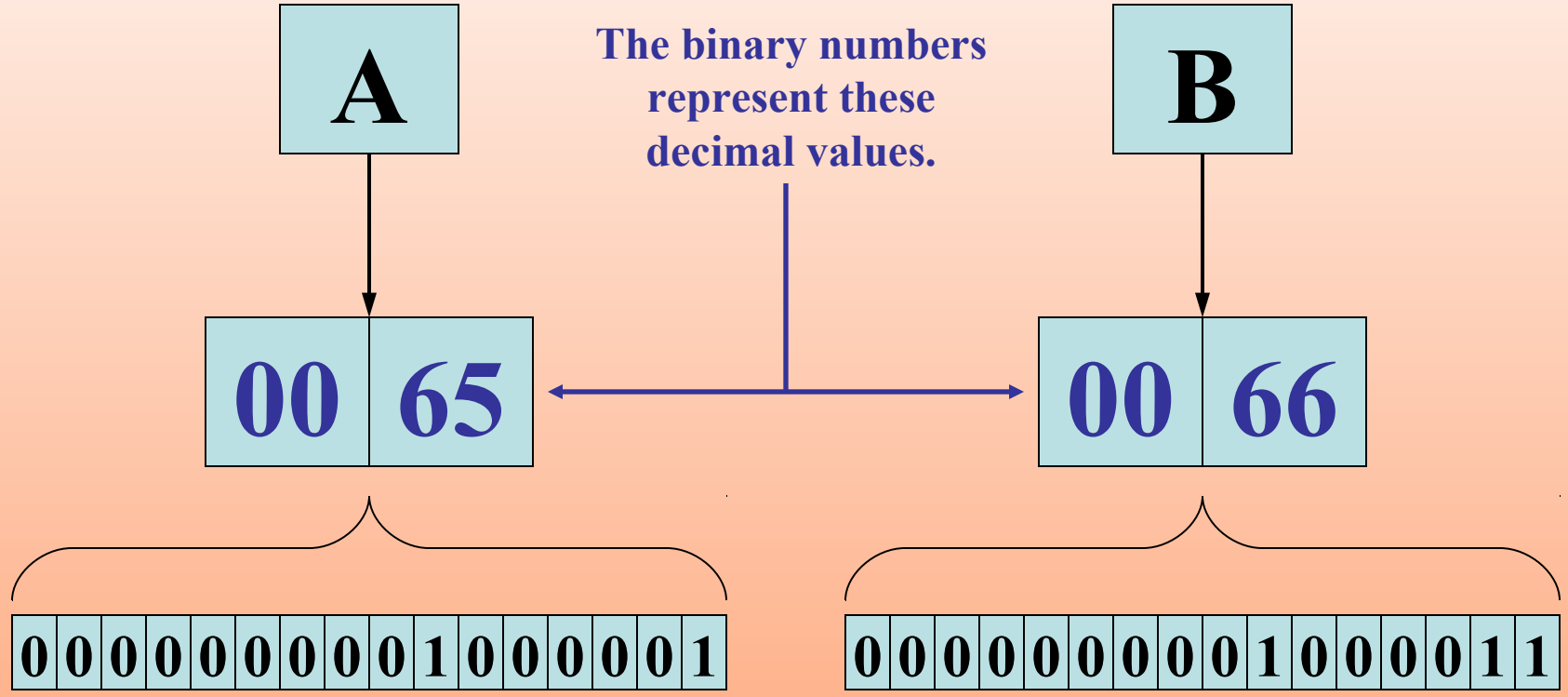
# Unicode



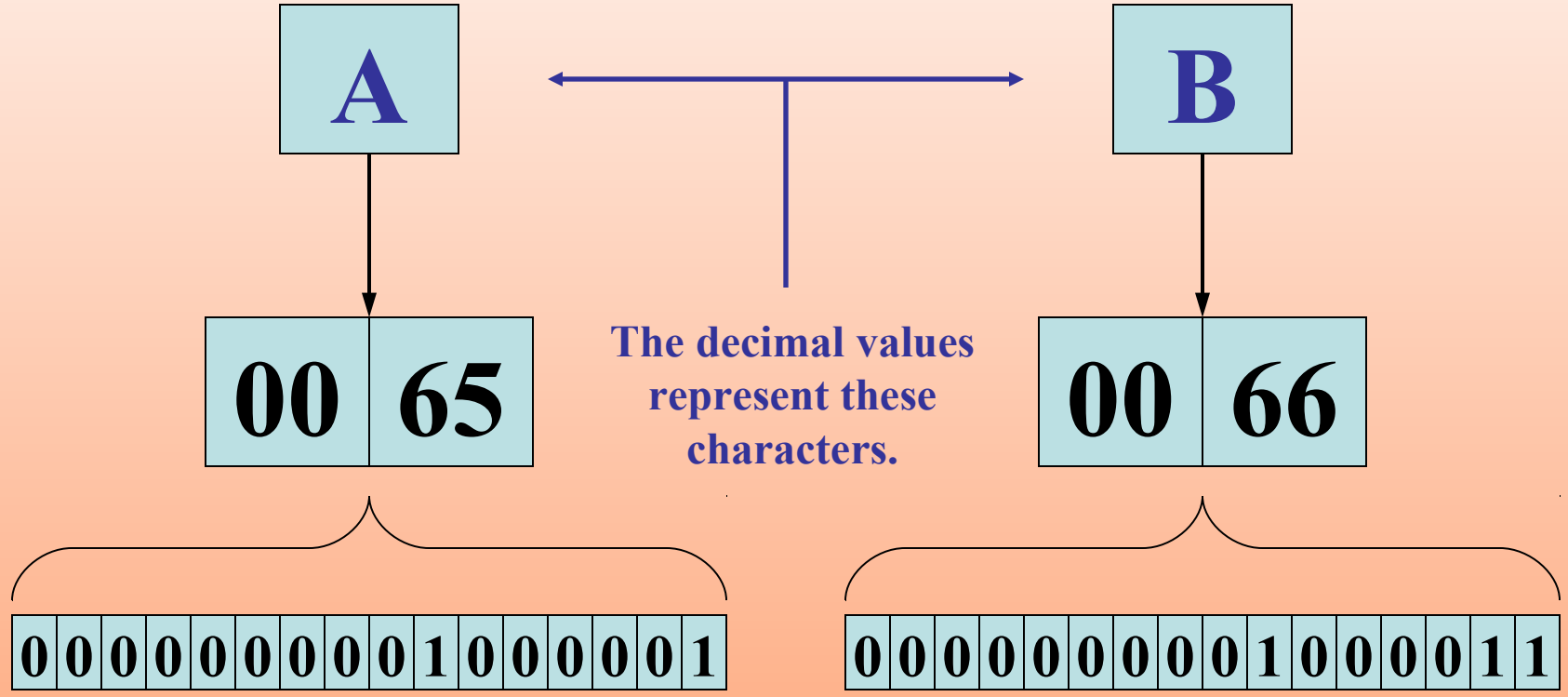
# Unicode


A
Characters are stored in memory as binary numbers.
B
00 65
00 66
0 0 0 0 0 0 0 0 0 1 0 0 0 0 0 1
0 0 0 0 0 0 0 0 0 1 0 0 0 0 1 1

# Unicode


A
The binary numbers represent these decimal values.
B
00 65
00 66
0 0 0 0 0 0 0 0 0 1 0 0 0 0 0 1
0 0 0 0 0 0 0 0 0 1 0 0 0 0 1 1

# Unicode


A
B
00 65
The decimal values represent these characters.
00 66
0 0 0 0 0 0 0 0 0 1 0 0 0 0 0 1
0 0 0 0 0 0 0 0 0 1 0 0 0 0 1 1

# Variable Assignment and Initialization

In order to store a value in a variable, an *assignment statement* must be used.

The *assignment operator* is the equal (=) sign.

The operand on the left side of the assignment operator must be a variable name.

The operand on the right side must be either a literal or expression that evaluates to a type that is compatible with the type of the variable.

# Variable Assignment and Initialization

```
// This program shows variable assignment.

public class Initialize
{
  public static void main(String[] args)
  {
    int month, days;

    month = 2;
    days = 28;
    System.out.println("Month " + month + " has " +
                       days + " Days.");
  }
}
```

**The variables must be declared before they can be used.**

# Variable Assignment and Initialization

```
// This program shows variable assignment.

public class Initialize
{
  public static void main(String[] args)
  {
    int month, days;

    month = 2;
    days = 28;
    System.out.println("Month " + month + " has " +
                       days + " Days.");
  }
}
```

**Once declared, they can then receive a value (initialization); however the value must be compatible with the variable's declared type.**

# Variable Assignment and Initialization

```
// This program shows variable assignment.

public class Initialize
{
  public static void main(String[] args)
  {
    int month, days;

    month = 2;
    days = 28;
    System.out.println("Month " + month + " has " +
                       days + " Days.");
  }
}
```

**After receiving a value, the variables can then be used in output statements or in other calculations.**

# Variable Assignment and Initialization

```
// This program shows variable initialization.

public class Initialize
{
  public static void main(String[] args)
  {
    int month = 2, days = 28;
    System.out.println("Month " + month + " has " +
                       days + " Days.");
  }
}
```

**Local variables can be declared and initialized on the same line.**

# Variable Assignment and Initialization

Variables can only hold one value at a time.

Local variables do not receive a default value.

Local variables must have a valid type in order to be used.

```
public static void main(String [] args)
 {
   int month, days; //No value given…
  System.out.println("Month " + month + " has " +
                          days + " Days.");
 }
```

**Trying to use uninitialized variables will generate a Syntax Error when the code is compiled.**

# Exercise

Can you write a program that have the following data stored in the program with initial values. You need to determine the data types and meaningful variable names. In the end display the values of these variables in the format of Variable name=value.

- My monthly salary of $102.12
- The number of CS majors 33
- Whether Paul will get an A from this class. He did get an A. So this variable's value is true.
- The letter grade of a student in CS121 is C
- The first sentence I said this morning, “Oh it's already eight!”