



420-101-VA: Programming 1

WEEK 1: JAVA FUNDAMENTALS

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** (Java is a case-sensitive language).
- Each Java class can be separated into parts.

```
// This is a simple Java program.  
  
public class Simple  
{  
    public static void main(String[] args)  
    {  
        System.out.println("Programming is great fun!");  
    }  
}
```

Analyzing The Example (1 of 3)

```
// A simple Java program.
```

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

```
public class Simple  
{
```

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 (2 of 3)


```
// 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 (3 of 3)

```
// A simple Java program.
```

```
public class Simple
{
    public static void main(String [] args)
    {
        System.out.println("Programming is great fun!");
    }
}
```



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

Parts of a Java Program

- Comments
 - Comments should be included to explain the purpose of the program and describe processing steps
 - 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.
 - For every left brace, or opening brace, there must be a corresponding right brace, or closing brace.

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 (1 of 2)

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

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

 **object**  **method name**  **information provided to the method (parameters)**

- This is because it is the only Java statement in the program.
- The rest of the code is either a comment or other Java framework code.
- The `System.out` object represents a destination (the monitor screen) to which we can send output

Java Statements (2 of 2)

- 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.

The println Method

- The `System.out` object represents a destination (the monitor screen) to which we can send output
- The `print` method is similar to the `println` method, except that it does not advance to the next line
- Therefore anything printed after a `print` statement will appear on the same line

Special Characters

//	double slash	Marks the beginning of a single line comment.
()	open and close parenthesis	Used in a method header to mark the parameter list .
{ }	open and close curly braces	Encloses a group of statements, such as the contents of a class or a method.
" "	quotation marks	Encloses a string of characters, such as a message that is to be printed on the screen
;	semi-colon	Marks the end of a complete programming statement

Escape Sequences

- What if we wanted to print the quote character?
- The following line would confuse the compiler because it would interpret the second quote as the end of the string

```
System.out.println("I said "Hello" to you.");
```

- An ***escape sequence*** is a series of characters that represents a special character
- An escape sequence begins with a backslash character (\)

```
System.out.println("I said \"Hello\" to you.");
```

Java Escape Sequences (1 of 2)

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

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

Java Escape Sequences (2 of 2)

- 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.

Variables and Assignment (1 of 3)

- A variable is a named storage location in the computer's memory.
- Programmers determine the number and type of variables a program will need.
- **Example:**

```
public class Variable
{
    public static void main(String[] args)
    {
        int value;
        value = 5;
        System.out.print("The value is ");
        System.out.println(value);
    }
}
```

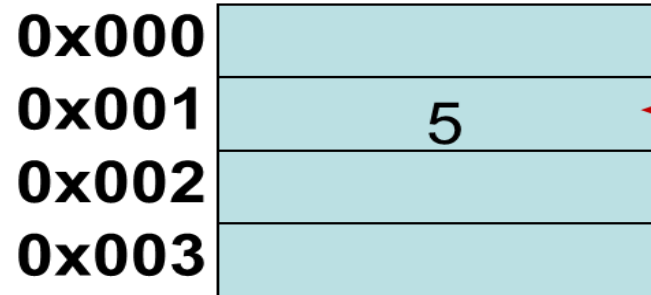
Variables and Assignment (2 of 3)

- Variables are used to store data. Every Java variable has a:
 - **Name:** chosen by the programmer
 - **Type:** specified in the declaration of the variable
 - **Size:** determined by the type
 - **Value:** the data stored in the variable's memory location

Variables and Assignment (3 of 3)

This line is called
a *variable declaration*.
`int value;`

The following line is known
as an assignment statement.
`value = 5;`



The value 5
is stored in
memory.

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

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

The integer 5 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 (1 of 3)

- 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 (2 of 3)

- 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 (3 of 3)

- 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.

Quick Check

What output is produced by the following?

```
System.out.println("X: " + 25);  
System.out.println("Y: " + (15 + 50));  
System.out.println("Z: " + 300 + 50);
```

```
X: 25  
Y: 65  
Z: 30050
```

Identifiers (1 of 2)

- Identifiers are programmer-defined names for:
 - classes
 - variables
 - methods
- Identifiers may not be any of the Java reserved keywords.

Identifiers (2 of 2)

- 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.
 - *Reserved words* (keywords) cannot be used as identifiers

Java Reserved Keywords

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

Identifiers – Self-documenting

- Identifiers 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**.

Identifiers – Self-documenting

- Do not be afraid if the identifier is too long, the most important thing is to **keep your code understandable**
- Identifiers with more than one words should follow the **camel-case**:
 - Class: `XxxxXxxxXxx`
 - Variable or Method: `xxxxXxxxXxxx`
 - For example: `classAvgScore`, `calcFinalScore`, `isUserStatusValid`
- Short abbreviations are also widely used for identifiers:
 - str: string
 - num: number
 - avg: average
 - idx: index
 - calc: calculate

Java Naming Conventions

- Variable names should begin with a lowercase 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.

Constants

- A *constant* is an identifier that is similar to a variable except that it holds the same value during its entire existence
- As the name implies, it is constant, not variable
- The compiler will issue an error if you try to change the value of a constant
- In Java, we use the `final` modifier to declare a constant

```
final int MIN_HEIGHT = 69;
```

Constants

- Constants are useful for three important reasons
 - First, they give meaning to otherwise unclear literal values
 - Example: `MAX_LOAD` means more than the literal 250
 - Second, they facilitate program maintenance
 - If a constant is used in multiple places, its value need only be set in one place
 - Third, they formally establish that a value should not change, avoiding inadvertent errors by other programmers

Primitive Data Types

- Primitive data types are built into the Java language and are not derived from classes.
- Java is a **static type programming language**, that means it is very strict about data types. (Python or JavaScript are **dynamic type programming language**, which is not that strict about data types)
- In Java, **putting a value of a data type into another data type variable may create a problem.**
- 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 $\pm 3.410 \times 10^{-38}$ to $\pm 3.41038 \times 10^{38}$, with 7 digits of accuracy
double	8 bytes	Floating-point numbers in the range of $\pm 1.710 \times 10^{-308}$ to $\pm 1.710308 \times 10^{308}$, with 15 digits of accuracy

Variable Declarations

- Variable Declarations take the following form:

DataType VariableName;

- `byte inches;`
- `short month;`
- `int speed;`
- `long timeStamp;`
- `float salesCommission;`
- `double distance;`

Integer Data Types

- `byte`, `short`, `int`, and `long` are all integer data types.
- They can hold whole numbers such as 5, 10, 23, 89, etc.
- Integer data types cannot hold numbers that have a decimal point in them.
- Integers embedded into Java source code are called **integer literals**.

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 (1 of 3)

- 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.

Example: Sale.java

```
// This program demonstrates the double data type.

public class Sale
{
    public static void main(String[] args)
    {
        double price, tax, total;

        price = 29.75;
        tax = 1.76;
        total = 31.51;
        System.out.println("The price of the item " +
                           "is " + price);
        System.out.println("The tax is " + tax);
        System.out.println("The total is " + total);
    }
}
```

Floating Point Literals (2 of 3)

- 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 (3 of 3)

- Literals cannot contain embedded currency symbols or commas.
 - `grossPay = $1,257.00; // ERROR!`
 - `grossPay = 1257.00; // Correct.`

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.

```
public class TrueFalse
{
    public static void main(String[] args)
    {
        boolean bool;
        bool = true;
        System.out.println(bool);
        bool = false;
        System.out.println(bool);
    }
}
```


The char Data Type

- The Java `char` data type provides access to single characters.
- `char` literals are enclosed in single quote marks.
 - `'a', 'Z', '\n', '1'`
- Don't confuse `char` literals with string literals.
 - `char` literals are enclosed in single quotes.
 - String literals are enclosed in double quotes.

```
public static void main(String[] args)
{
    char letter;
    letter = 'A';
    System.out.println(letter);
    letter = 'B';
    System.out.println(letter);
}
```

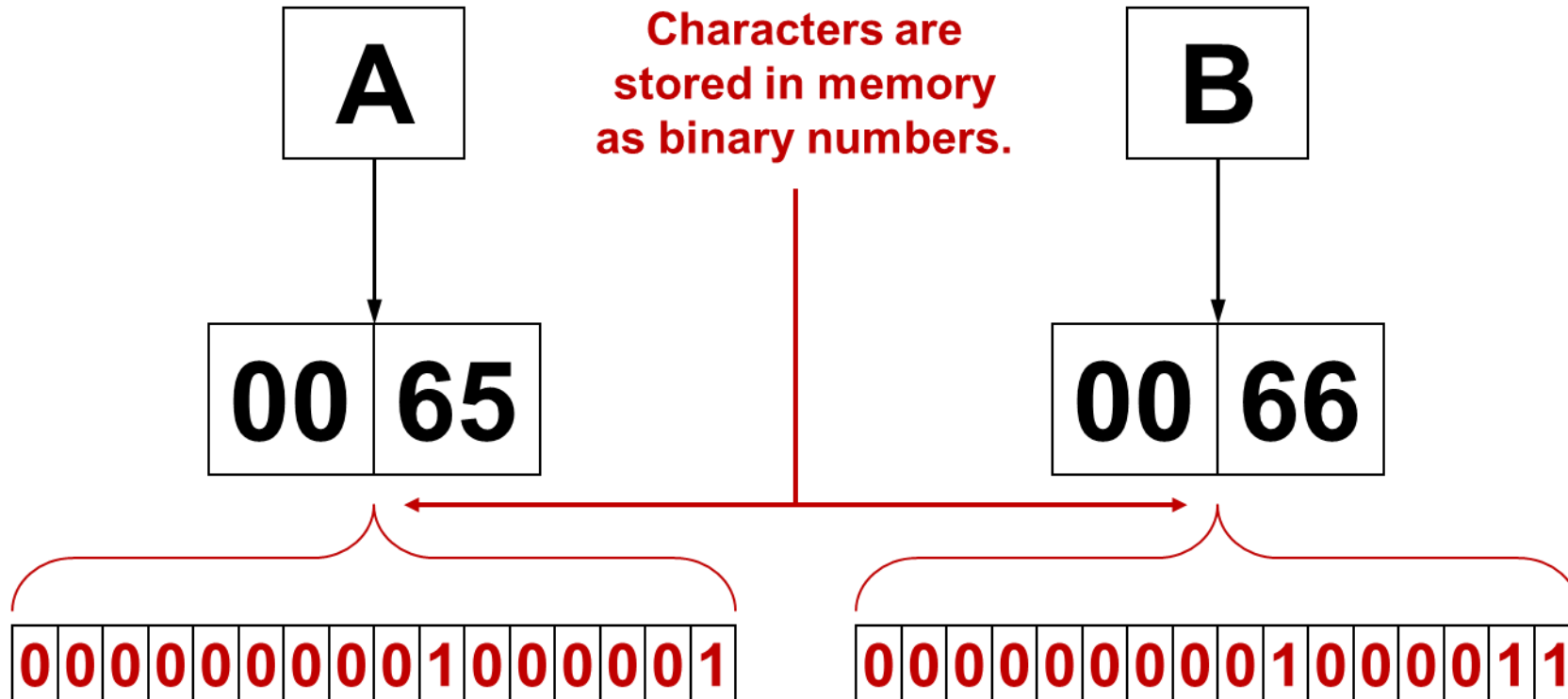
The `char` Data Type

- Computers see characters as binary numbers, so each character is encoded then stored in memory
- Encoding is mapping each character to its binary representation with the help of encoding scheme
 - Character **to** Unicode **to** binary
- A `char` variable in Java can store any character from the *Unicode character set*
- The Unicode character set uses sixteen bits per character, allowing for 65,536 unique characters
- It is an international character set, containing symbols and characters from many world languages
 - it as a dictionary which defines what number should be stored for each character (English, French, Arabic, Korean, Japanese, Chinese, etc.).

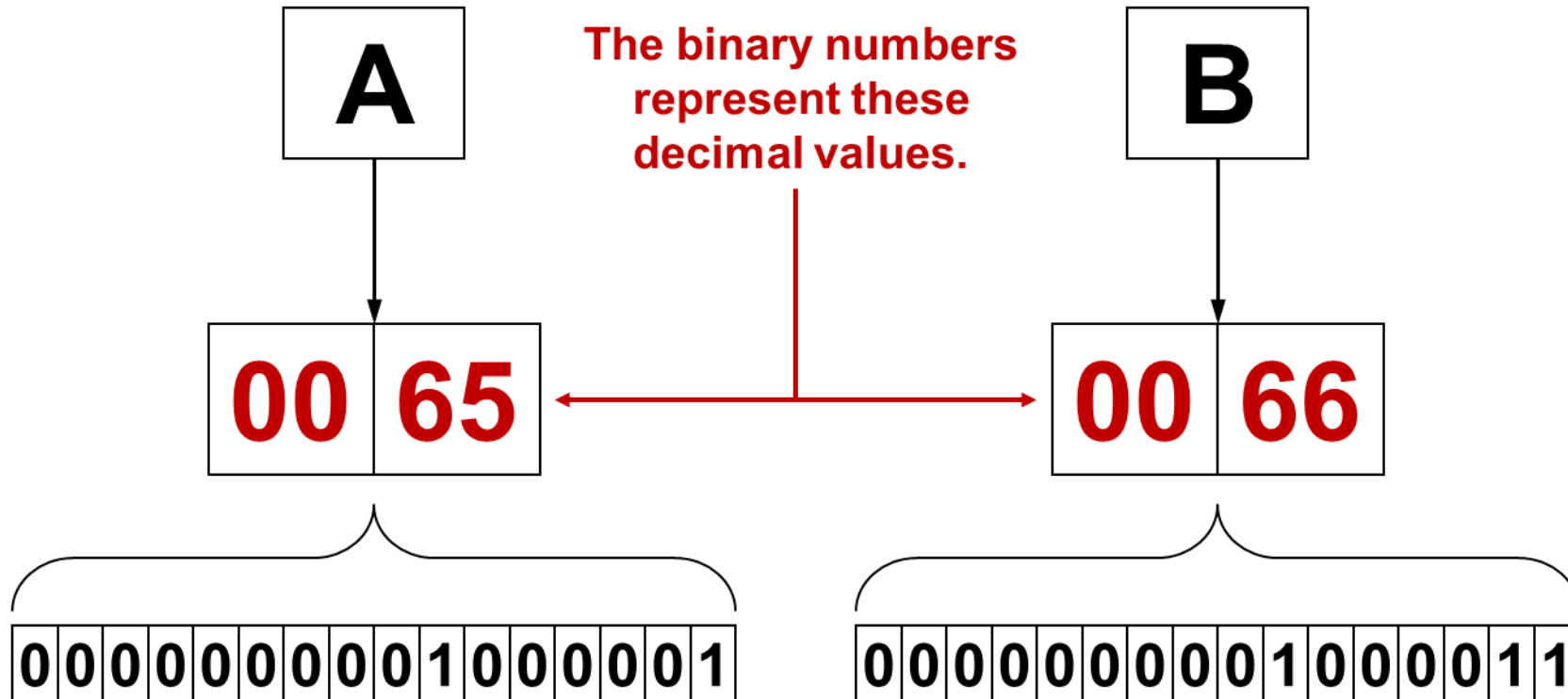
Unicode (1 of 4)

- 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.

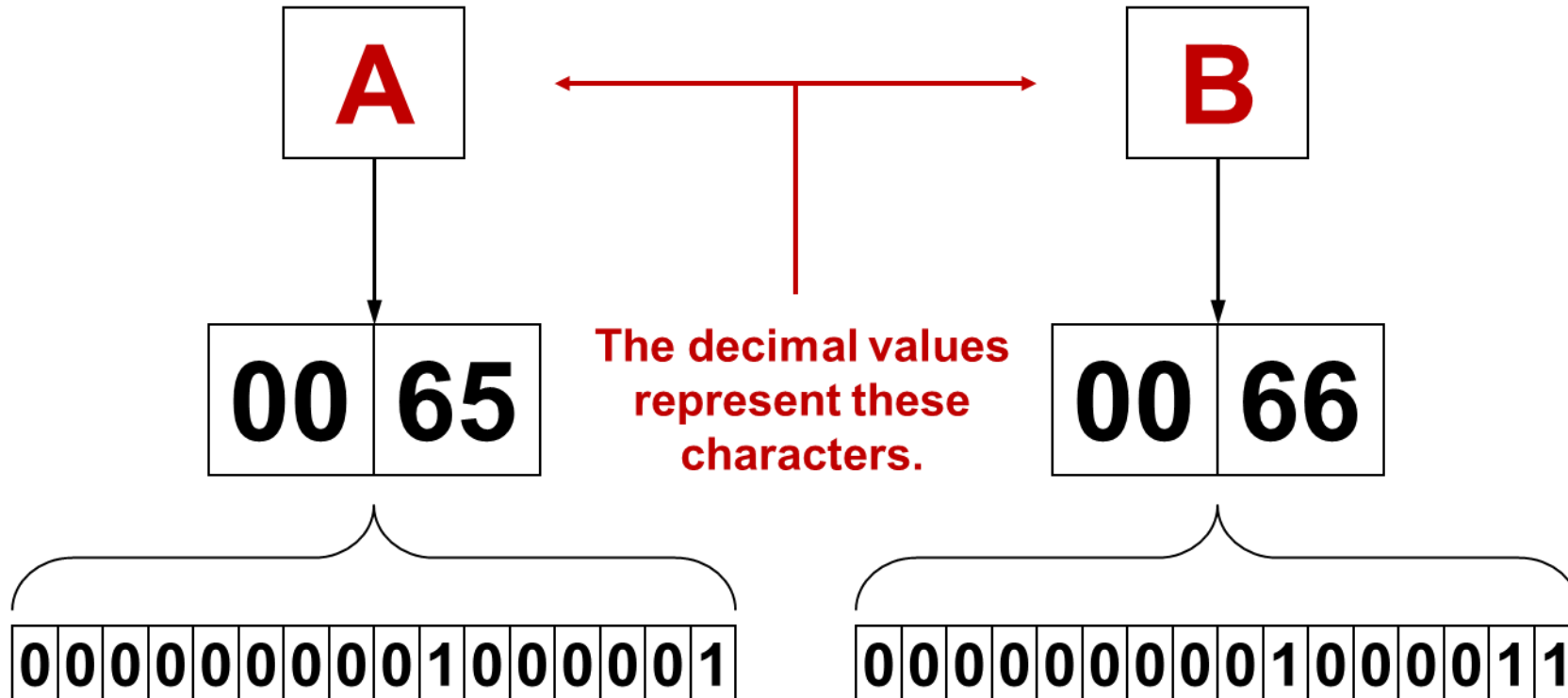
Unicode (2 of 4)



Unicode (3 of 4)



Unicode (4 of 4)



Variable Assignment and Initialization (1 of 6)

- 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 (2 of 6)

```
// 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 (3 of 6)

```
// 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 (4 of 6)

```
// 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 (5 of 6)

```
// 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 (6 of 6)

- 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.