Java programming for C/C++ developers Part I

Lecture 2

Dr. Tamer ABUHMED

Java Programming Course (SWE2023)

College of Computing

SUNG KYUN KWAN UNIVERSITY

Outline

- Overview about the similarities between C++ and Java
- Java data types vs. C++ data types
- C++ vs. Java Programming Style
- Common differences between C++ and Java
  - Variables
  - Constants
  - Strings
  - Arrays
  - Multidimensional Arrays
- Common differences in Syntax and Semantics

Some Similarities between C++ and Java

- Simple (primitive) types: int, double, char
- Control Structures if-else, switch, while, for
- Arithmetic expressions
- Both have a string type: C++ string, Java String.
- Arrays
- Both have classes.
- Both have a "main".

Java data types and their C++ counterparts

- Java
  - byte (signed, 8 bits)
  - short (signed, 16 bits)
  - int (signed, 32 bits)
  - long (signed, 64 bits)
  - boolean (true/false)
  - char (16 bits, Unicode)
  - float (32 bits)
  - double (64 bits)
  - void
  - String
- C++
  - char (sort of)
  - int, short
  - long, int
  - long
  - bool
  - char (sort of - 8 bit ASCII)
  - float
  - double
  - void
  - string

Note: string type is not primitive, built-in type in either language

C++ vs. Java Programming Style

- C++ and Java divide a program into pieces (for separate compilation) in different ways.
- C++: Traditionally has
  - an interface (header) file.h
  - implementation file(s).cpp
  - application (driver) file.dll
  - C++: Can confine a program to a single file if you want.
- Java: A compilation unit is always a class definition.
  - Every class is in a separate file (except for some special cases).
  - No header files.
  - Normally, you have no one file programs in Java.

Differences between C++ and Java

- Java DOESN'T have
  - **Multiple inheritance** (well, it somewhat does through interfaces and, but this is not a true inheritance).
  - **Templates**. Vast majority of objects inherit from the Object class or its descendants (implicitly or explicitly), so any object can be cast to the Object class.
  - **Pointers**, only references. All objects manipulated by reference by default in Java, not like in C++. So, there is no & in the syntax for function parameters.
  - **Operator overloading**
  - **Destructors**
  - **Delete** operator

Templates

More Differences

- Java has **automatic garbage collection**.
  C++ does not.
- C++ says "function". Java says "method".
- In Java, every **method is a member of some class**.
  You cannot have a freestanding (global) function in Java.
- C++ has built in **console I/O**. Java has no standard console input (but does have standard console output.)

More Differences

- C++ has pointer types.
  - Java has no pointer types .
- Assignment (=) and equality comparison (==) have minor differences.
- C++ gives a choice of parameter types but Java has no choice of parameter types.
  - C++ and Java: Call-by-value
    - `void f(int n);`
  - C++: Call-by-reference
    - `void f(int& n);`
  - Other C++ variants:
    - `void f(const int& n);`
    - `void f(const int n);`

Variables

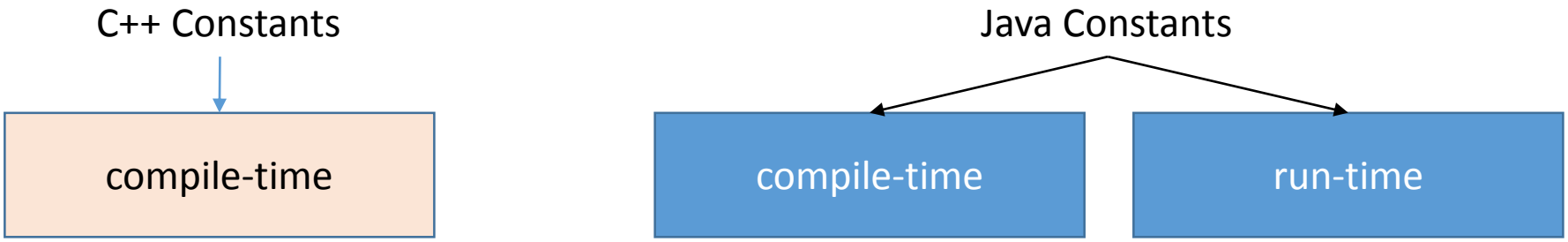
- Variables are declared in the same way in C++ and Java.
- Assigning values to variables is done with the same syntax in both languages.
- C++ also supports an alternative way of initializing variables called constructor initialization.
- Unlike C++, all variables must be contained within a type or a function in Java.
- Java do not allow local variables to be used unless they are initialized.

C++	Java

`int a = 50, b = 10;`
`int b(50); // constructor initialization`
`int *p_int = new int;`
`delete p_int; //delete variable`
	`int a = 50, b = 10;`
	
	`//Memory Allocation is automated (NO`
	`//NEED) to delete variable after use`

Constants

Compile-time constants are replaced by the compiler and must therefore be initialized
to a constant value at the same time as they are declared.
Run-time constants on the other hand are not set until the program runs.



C++

```
#define PI 3.14

const int PI = 3.14;
```

Java

```
final double E = 2.72; // run-time constant
final static double C = 512; // compile-time constant
final static int RND = (new
java.util.Random()).nextInt(); // run-time constant
```

Strings

- Java support **String** variables to store values that contain string literals.
- In C++, **string** header from the standard library needs to be included in order to work with strings.
- Both perform string concatenation using the ‘+’ or ‘=+’ operations.
- C++ strings can be modified, whereas Java strings are immutable.

C++	Java

`#include <string>`
`using namespace std;`
`string a = "Hello";`
`string b(" World");`
`string c = a + b;`
	`String a = "Hello";`
	`String b =`
	`new String(" World");`
	`String c = a + b;`

Arrays

- Unlike C++, Java arrays are objects that are bounds-checked.
- In java, the arrays are objects and have methods for retrieving their own length, but in C++, it not.

C++

```
int x[3]; //stack-based array
int y[ ] = {1,2,3}; //stack-based array
int z[ ] = new int[3]; //heap-based array
delete z[ ]; //delete array
int values[ 10 ];
values[ 20 ] = 2; //it just writes the data there
```

Java


```
int x[ ] = new int[3];
int[ ] y = {1,2,3}; //initializing array
//=========================
int[ ] values = new int[ 10 ];
values[ 20 ] = 2; // error! You get an
//ArrayIndexOutOfBoundsException
```

Multidimensional Arrays

- Java supports jagged arrays but C++ does not.
- C++ creates rectangular arrays using the same syntax as Java's jagged arrays but the dimensions must be specified, which is optional in Java.

C++	Java
#include <string> std::string r[2][2] = {{ "00", "01" }, { "10", "11" }};	String[][] j = {{ "00" }, { "10", "11" }};

r

00	01
10	11

rectangular array

j

00	
10	11

jagged array

Input/Output console

- C++ has built in console input and output. Java has no standard console input (but does have standard console output.)
- C++: has cin, cout, cerr
- Java has:

  `System.out.print` and

  `System.out.println`

  but NO console input.
- AP does not require console input.
- For Input concole: there are classes for console input that are not part of Java standard but written in Java:

  e.g., **`java.io.Console`**
- **`java.util.Scanner`**

A Sample Java Class

Source

```
public class PetRecord
{
   private String name;
   private int age;//in years

   public PetRecord(String initName, int initAge)
   {
     name = initName;
     if ((initAge < 0))
        System.out.println("Error");
     else
        age = initAge;
}
```

A Sample Java Class

Source

```
 public void writeOutput()
   {
      System.out.println("Name: "+ name);
      System.out.println("Age: " + age + " years");
   }

public static void main(String[] a){
PetRecord X = new PetRecord("Loli",2);
X.writeOutput();
}
}
```

Output

```
Name: Loli
Age: 2 years
```

Syntax and Semantics

Syntax and Semantics: Main Method

- The main method is the method from which the application execution starts.
- In Java, main method must have the public static void modifiers and accept the command-line arguments as a string array.
- The main method in C++ needs to have the int return type, but it is optional whether the method actually returns an integer or not.

C++	Java
int main(int argc, const char* a[])	public static void main(String[] a)
int main (int argc, char ** argv)	public static void main(String a[])
int main()	public static void main(String... a)

Syntax and Semantics: Conditions

- In Java if-statement, the conditional expression inside the parenthesis must evaluate to a Boolean value (true or false)
- Conditional expression in C++ can be anything that evaluates to a number (0 means false, number means true)

C++	Java
if(expression) {}	if(bool) {}
else if(expression) {}	else if(bool) {}
else {}	else {}

Syntax and Semantics: Jump Statements

- Jump statements redirect the program's execution from one program location to another. Java and C++ have the following four jump statements: break, continue, return, and throw.
- C++ also support the goto statement

C++	Java

while (true)
{ while (true)
{ goto MyLabel; }
}
MyLabel:
	while (true)
	{ while (true)
	{ break MyLabel; }
	}
	MyLabel:

Syntax and Semantics: Passing Arguments

- Usually arguments to a method can be passed either by value or by reference.
- **Pass by reference:** both value and reference data types can be changed or replaced and the changes will affect the original reference.
- **Pass by value variables** will have a copy from the original reference.
- For passing parameters C++ support both pass-by-reference and pass-by-value, while Java supports only pass-by-value.

Syntax and Semantics: Passing Arguments

C++

```
int ByValue(int a) {}
int ByRef(int &a) {}
int ByPointer(int* a) {}
 int x = 1;


int main(int argc, char *argv[]){
ByValue(x);
ByRef(x);
ByPointer(&x);
}
```

Java

```
Class Example{
void ByValue(int a) {}
 int x = 1;


public static void main(String [] arg){
ByValue(x);
}
}
```

Syntax and Semantics: Method Overloading

- Both C++ and Java support method overloading which means that a function can be defined multiple times with different arguments.
- Default parameters are also available in C++, but not in Java
- You cannot overload (redefine) = and == in Java

C++	Java
`void F(int a = 3) {}` `void F(float a = 3.14) {}`	`void F(int a) {}` `void F(float a) {}`

Syntax and Semantics: Variable Parameter Lists

- Both Java and C++ support passing a variable number of arguments to methods.
- However, variable parameter in C++ must be as an argument to the end of the list

C++	Java

```
#include <stdargs.h>
void F(int first, ...) {
 int i = first;
 va_list marker; // retrieve arguments
 va_start(marker, first);
  while(i != -1)
 i = va_arg(marker, int); va_end(marker);}
```

```
void F(int... args)
{
 for(int i : args) {}
}
```

Syntax and Semantics: Functions

- Java functions must always belong to a class, whereas in C++ functions may also be declared globally.
- Unlike Java, Functions as well as classes in C++ need to be declared before they can be called.
- In C++, If the function is to be used before it is implemented, then a prototype must be specified (forward declarations).

C++

```
void MyMethod(int x); // prototype
void MyMethod(int x) {}
```

Java

```
class MyClass
{
 void MyMethod(int x) {}
}
```

Syntax and Semantics: Libraries

- To organize codes into different **namespaces** (libraries or packages) and reuse these codes in any future software.

C++	Java
#include <string> using namespace std; //references to classes from a library	import java.lang.String; //references to classes from the Java class library

Summary

- We explained the common similarities between C++ and Java
- A comparison between the Java data types and C++ data types was shown.
- C++ vs. Java Programming Style
- Common differences between C++ and Java
  - Variables
  - Constants
  - Strings
  - Arrays
  - Multidimensional Arrays
- Common differences in Syntax and Semantics