CSE341 Programing Language

HW#1

Berkan AKIN

171044073

Lisp Versus Java

This is a comparison of java programming languages and lisp. It will be learned from the printer models of programming languages, and it will touch on many interests such as the first. Between the two programming languages, the choice is made between good and bad. object oriented programming and functional programming features will be compared

Lisp Programming Language

Lisp is a **functional programming** language with imperative features. By functional we mean that the overall style of the language is organized primarily around expressions and functions rather than statements and subroutines. Every Lisp expression returns some value. Lisp is the second oldest high-level programming language after Fortran and has been of great diversity since its first and a number of dialects through its history. The most "general-purpose Lisp dialects today are Common Lisp and Scheme. Lisp was invented by John McCarthy while at the Massachusetts Institute of Technology (MIT) in 1958. LISP expressions are called symbolic expressions or s expressions. S statements consist of three valid objects, atom, list, and string. Any s statement is a valid program. Lisp programs run on an interpreter or as compiled code.

The interpreter checks the source code in an iterative loop, also called the read-evaluation-write cycle (REPL). Reads the program code, evaluates it, and prints the values returned by the program

In this comparison, the usage feature of the lisp language will be compared.

Java Programming Language

Java is a general-purpose, class-based**, object-oriented programming** language for having fewer applications. It is an information platform for application development. This advanced Java is fast, safe and secure. The processor is used in computers, data, game consoles, supercomputers, mobile phone, etc. It is widely used to develop Java applications. It is considered one of the fast, secure and reliable programming languages most organizations prefer to build their projects. It is considered one of the fast, secure and reliable programming languages most organizations prefer to build their projects.

The syntax of the Lisp programming language

LISP programs consist of three basic building blocks.

- Atom
- List
- String

Atom is a number or string of adjacent characters. Contains numbers and special characters.

Example

Apple

Banana#45

Kocaeli-gebze

List is an array of parenthesized atoms and/or other lists.

Example

```
( i am a list)
(a ( a b c) d e fgh)
( )
```

String is a group of characters enclosed in double quotes.

```
" I am a string"
"Hello Gebze Teknik! "
```

The semicolon symbol (;) is used to indicate the comment line.

```
(write (+ 3 4 5)) ;  this is adddition
```

Some of the **important points** to note are:

- Basic numeric operations in Lisp +, -, *, and /
- LiSP represents the function call f(x) as (f x), for example sin (45) is written as sin 45
- LISP expressions are case insensitive, sin 45 or SIN 45 are the same.
- Lisp tries to evaluate everything, including the arguments of a function. Only three types of items are fixed and always return their own values

Variables

In Lisp, each variable is represented by a symbol. The name of the variable is the name of the symbol and is stored in the symbol's storage cell

Global Variables

Global variables have persistent values throughout the LISP system and remain in effect until a new value is specified. Global variables are usually declared using the defvar construct.

Example

```
(defvar x 234)
(write x)
```

Because Lisp has no type declaration for variables, you directly specify a value for a symbol with the **setq** construct.

Example

```
(setq x 10)
```

Local Variables

Local variables are defined within a particular procedure. Parameters called arguments in the function definition are also local variables. Local variables can only be accessed within the corresponding function. Like global variables, local variables can be created using the setq construct. There are two more constructs for creating local variables - let and prog.

The let construct has the following syntax.

```
(let ((x 'a) (y 'b)(z 'c))
(format t "x = ~a y = ~a z = ~a" x y z))
```

Output

```
x = A y = B z = C
```

The prog structure also includes a list of local variables as its first argument; followed by the prog body and any number of s statements.

The prog function sequentially executes the list of s statements and returns zero unless encountering a function call called return. The argument of the return function is then evaluated and returned.

Example

```
(prog ((x '(a b c))(y '(1 2 3))(z '(p q 10)))
(format t "x = ~a y = ~a z = ~a" x y z))
```

Output

```
x = (A B C) y = (1 2 3) z = (P Q 10)
```

Operators

+	Adds two operands	(+A B) will give 30
-	Subtracts second operand from the first	(- A B) will give -10
*	Multiplies both operands	(* A B) will give 200
/	Divides numerator by de-numerator	(/ B A) will give 2
mod,rem	Modulus Operator and remainder of after an integer division	(mod B A)will give 0

Comparation Operators

=	Checks if the values of the operands are all equal or not, if yes then condition becomes true.	(= A B) is not true.
/=	Checks if the values of the operands are all different or not, if values are not equal then condition becomes true.	(/= A B) is true.
>	Checks if the values of the operands are monotonically decreasing.	(> A B) is not true.

<	Checks if the values of the operands are monotonically increasing.	(< A B) is true.

Logical Operations

and	It takes any number of arguments. The arguments are evaluated left to right. If all arguments evaluate to non-nil, then the value of the last argument is returned. Otherwise nil is returned.	(and A B) will return NIL.
or	It takes any number of arguments. The arguments are evaluated left to right until one evaluates to non-nil, in such case the argument value is returned, otherwise it returns **nil**.	(or A B) will return 5.
not	It takes one argument and returns **t** if the argument evaluates to **nil.**	(not A) will return T.

Conditions

The **cond** construct in LISP is most commonly used to permit branching.

Syntax for **cond** is –

example

```
(cond   (test1    action1)
  (test2    action2)
  ...
  (testn    actionn))
```

The **if** macro is followed by a test clause that evaluates to t or nil. If the test clause is evaluated to the t, then the action following the test clause is executed. If it is nil, then the next clause is evaluated.

Syntax for if –

Example

(if (test-clause) ($action_1$) ($action_2$))

The **when** macro is followed by a test clause that evaluates to t or nil. If the test clause is evaluated to nil, then no form is evaluated and nil is returned, however the test result is t, then the action following the test clause is executed.

Syntax for when macro –

example

(when (test-clause) (<$action_1$))

The **case** construct implements multiple test-action clauses like the **cond** construct. However, it evaluates a key form and allows multiple action clauses based on the evaluation of that key form.

The syntax for case macro is −

The template for CASE is

example

```
(case  (keyform)
((key1)   (action1   action2 ...) )
((key2)   (action1   action2 ...) )
...
((keyn)   (action1   action2 ...) ))
```

Array

LISP allows you to define single or multidimensional arrays using the make array function. An array can store any LISP object as its elements.All arrays consist of contiguous memory locations. The lowest address corresponds to the first item, and the highest address to the last item. The number of dimensions of an array is called its rank. In Lisp, an array element is denoted by an array of non-negative integer indices. The length of the array must be equal to the rank of the array. Indexing starts from scratch. For example, to create a 10-cell array named myarray, we could write.

Example

```
(setf my-array (make-array '(10)))

(write (setf my-array (make-array '(10))))
(terpri)
(setf (aref my-array 0) 25)
(setf (aref my-array 1) 23)
(setf (aref my-array 2) 45)
 (write my-array)
```

The syntax of the Java programming Language

When we think of a Java program, it can be defined as a collection of objects that communicate by calling each other's methods. Let's look briefly at what class, object, method and instance variables mean. Object - Objects have states and behaviors. Example A dog has states

- color, name, breed and behavior such as wagging its tail, barking, eating. An object is an instance of a class. Class
- A class can be defined as a template/drawing that describes the behavior/state supported by the object of its type. Methods
- A method is basically a behavior. A class can contain many methods. It is in the methods in which logic is written, data is manipulated, and all actions are executed. Instance Variables
- Each object has its own unique set of instance variables. An object's state is created by the values assigned to these instance variables.

Basic Syntax

About Java programs, it is very important to keep the following points in mind. Case Sensitivity - Java is case sensitive, which means that the identifier hello and hello will have different meanings in Java.

Class Names - The first letter for all class names must be Uppercase. If several words are used to form the name of the class, the first letter of each inner word must be Uppercase.

Example: class MyFirstJavaClass Method Names

- All method names must start with a lowercase letter. If several words are used to form the name of the method, the first letter of each inner word should be Uppercase.

Example: public void myMethodName()

Program File Name - The name of the program file must exactly match the class name. When saving the file, you need to save it using the class name (remember that Java is case sensitive) and add 'java' to the end of the name (if the filename and classname don't match, your program won't compile). However, please note that if you do not have a common class in the file, the file name may differ from the class name. It is also not mandatory to have a common class in the file. Example: Suppose the class name is 'MyFirstJavaProgram'. Then the file should be saved as 'MyFirstJavaProgram.java'

public static void main(String args[])

− Java program processing starts from the main() method, which is a mandatory part of every Java program.

Examples of illegal identifiers: 123 abc, -salary.

Java Modifiers

Like any other language you can have classes, methods, etc. It is also possible to change, using modifiers. There are two categories of modifiers −

Access Modifiers - default, public, protected, private

Java Datatypes

Variables are nothing but dedicated memory locations to store values. This means that when you create a variable, you allocate some memory. Based on the data type of a variable, the operating system allocates memory and decides what can be stored in the allocated memory. Therefore, by assigning different data types to variables, you can store integers, decimals or characters in these variables.

There are two data types in Java − Primitive Data Types ,Reference / Object Data Types

Primitive Data Types

There are eight primitive data types supported by Java. Primitive data types are predefined by the language and named with a keyword. Let's take a closer look at the eight primitive data types.

byte

- The byte data type is two complementary integers with an 8-bit signature
- Minimum value -128 (-2^7)
- The maximum value is 127 (inclusive) (2 ^ 7 -1)

Example:

byte a = 100, byte b = -50

short

- The short data type is two 16-bit signed integers
- Minimum value -32.768 (-2^15)
- The maximum value is 32,767 (inclusive) (2 ^ 15 -1)
  Example:
  short s = 10000, short r = -20000

integer

- The int data type is a 32-bit signed two-complement integer.
- Minimum value - 2,147,483,648 (-2 ^ 31)
- The maximum value is 2,147,483,647 (inclusive) (2 ^ 31 -1)
- Integer is usually used as the default data type for integral values unless there is a memory issue.

Example

int a = 100000, int b = -200000

long

- The long data type is two complementary integers with a 64-bit signature
- The minimum value is -9,223,372,036,854,775,808 (-2^63)
- Maximum value 9,223,372,036,854,775,807 (inclusive) (2 ^ 63 -1)
- This type is used when a larger range than int is required
- Default value is 0l

Example

long a = 100000L, long b =-200000L

float

- The floating point data type is single precision 32 bit IEEE 754 floating point
- Float is mainly used to save memory on large arrays of floating point numbers
- Float data type is never used for exact values such as currency

Example

float f1 = 234.5f

douple

- double data type is double precision 64 bit IEEE 754 floating point
- This data type is often used as the default data type for decimal values, usually the default selection
- Double data type should never be used for exact values such as currency

Example:

```
double d1 = 123.4
```

boolean

- boolean data type represents a bit of information
- There are only two possible values: true and false.
- This data type is used for simple flags that follow true/false conditions
- Default value is false
- Example: boolean one = true
- The char data type is a single 16-bit Unicode character

Example

```
char letterA = 'A'
```

Java Variables

Here are the variable types in Java −

- Local Variables
- Class Variables (Static Variables)
- Instance Variables (Non-Static Variables)

Out of Access Modifiers - final, abstract, strictfp

Example

```
int a, b, c;
int a = 10, b = 10;
byte B = 22;
double pi = 3.14159;
char a = 'a';
```

Local Variables

Local variables are declared in methods, constructors, or blocks. Local variables are created when the method, constructor, or block is entered, and destroyed after the variable exits the method, constructor, or block.

Example

```
public class Test {
  public void pupAge() {
    int age = 0;
```

```
    age = age + 7;
    System.out.println("Puppy age is : " + age);
  }

  public static void main(String args[]) {
    Test test = new Test();
    test.pupAge();
  }
}
```

Instance Variables

Instance variables are declared in a class, but outside a method, constructor or any block. When a space is allocated for an object in the heap, a slot for each instance variable value is created. Instance variables are created when an object is created with the use of the keyword 'new' and destroyed when the object is destroyed.

Example

```
public class Employee {

  // this instance variable is visible for any child class.
  public String name;

  // salary  variable is visible in Employee class only.
  private double salary;

  // The name variable is assigned in the constructor.
  public Employee (String empName) {
    name = empName;
  }

  // The salary variable is assigned a value.
  public void setSalary(double empSal) {
    salary = empSal;
  }

  // This method prints the employee details.
  public void printEmp() {
    System.out.println("name  : " + name );
    System.out.println("salary :" + salary);
  }

  public static void main(String args[]) {
    Employee empOne = new Employee("Ransika");
    empOne.setSalary(1000);
    empOne.printEmp();
  }
}
```

Class/Static Variables

Class variables also known as static variables are declared with the static keyword in a class, but outside a method, constructor or a block.

There would only be one copy of each class variable per class, regardless of how many objects are created from it.

Example

```
public class Employee {

  private static double salary;

  public static final String DEPARTMENT = "Development ";


 public static void main(String args[]) {

    salary = 1000;

    System.out.println(DEPARTMENT + "average salary:" + salary);

  }

}
```

Arithmetic Operators

asd

+ (Addition)	Adds values on either side of the operator.	A + B will give 30
- (Subtraction)	Subtracts right-hand operand from left-hand operand.	A - B will give -10
* (Multiplication)	Multiplies values on either side of the operator.	A * B will give 200
/ (Division)	Divides left-hand operand by right-hand operand.	B / A will give 2
% (Modulus)	Divides left-hand operand by right-hand operand and returns remainder.	B % A will give 0
++ (Increment)	Increases the value of operand by 1.	B++ gives 21
-- (Decrement)	Decreases the value of operand by 1.	B-- gives 19

Relational Operators

== (equal to)	Checks if the values of two operands are equal or not, if yes then condition becomes true.	(A == B) is not true.
!= (not equal to)	Checks if the values of two operands are equal or not, if values are not equal then condition becomes true.	(A != B) is true.
> (greater than)	Checks if the value of left operand is greater than the value of right operand, if yes then condition becomes true.	(A > B) is not true.
< (less than)	Checks if the value of left operand is less than the value of right operand, if yes then condition becomes true.	(A < B) is true.

Loop

While Loop

Repeats a statement or group of statements while a given condition is true. It tests the condition before executing the loop body.

For Loop

Execute a sequence of statements multiple times and abbreviates the code that manages the loop variable.

Do… while loop Like a while statement, except that it tests the condition at the end of the loop body.

Decision Making

if...else statement

An if statement can be followed by an optional else statement, which executes when the boolean expression is false.

switch statement

A switch statement allows a variable to be tested for equality against a list of values.

Declaring Array Variables

To use an array in a program, you must declare a variable to reference the array, and you must specify the type of array the variable can reference. Here is the syntax for declaring an array variable –

example

```
dataType[] arrayRefVar;
```

or

dataType arrayRefVar[];

Lisp and Java comparison

• **Readability**

Java programming language is more readable than lisp programming language

• **Writability**

Java is easier to write than lisp

• **Simplicity, orthogonality**

The features of the Lispte language are simpler than Java. The features of the same task in the language do not overlap too much. It intersects a lot in the Java programming language. As a result, Lisp is a simpler and more orthogonal Java programming language.

• High expressive power, flexibility

Stricter in Java type checking. Lisp is more flexible in this regard. In general, lisp programming language has a more flexible structure than java.

• **Reliability**

Java is a more reliable language than lisp because it has type checking.

• **Cost (influenced by above)**

Lisp is a more difficult language to learn. Writing and readability of the language is more difficult than Java. There is more documentation for learning Java. These make it easier to train and find a developer. For this reason Java is less costly than lisp

• **Maintenance**

Maintaining the java programming language becomes easy with the right software design since it is an object-oriented programming language. Since Lisp is a difficult language to read and write in programming language, it makes maintenance difficult.