CSCI 410

16 – The Jack Language

Credit
This lecture is modified from one at www.nand2tetris.org

The Jack Programming Language

Why Study Jack

We use Jack as a platform for teaching:

• How to build a compiler
• How the compiler and the language interface with the operating system
• How the topmost piece in the software hierarchy fits into the big picture

Jack can be learned (and un-learned) in one hour.

Hello world

```java
/** Hello world program. */
class Main {
    function void main () {
        // Prints some text using the standard library
        do Output.printString("Hello World");
        do Output.println(); // New line
        return;
    }
}
```

Some observations:

- Java-like syntax
- Typical comments format
- Standard library
- Language-specific peculiarities.

Example: Procedural Programming

Jack program = a collection of one or more classes.

There is always a Main class.

Execution begins in Main.main().

```java
/** Sums up 1 + 2 + 3 + ... + n */
class Main {
    function int sum (int n) {
        var int sum, i;
        let sum = 0;
        let i = 1;
        while (~i > n) {
            let sum = sum + i;
            let i = i + 1;
        }
        return sum;
    }

    function void main () {
        var int n;
        let n = Keyboard.readInt("Enter n: ");
        do Output.printString("The result is: ");
        do Output.printInt(sum(n));
        return;
    }
}
```
Example: Procedural Programming

```java
class Main {
    /** Sums up 1 + 2 + 3 + ... + n */
    function int sum (int n) {
        var int sum = 1;
        let sum = sum + 1;
        while (<n > n) {
            let sum = sum + 1;
            let i = i + 1;
        }
        return sum;
    }
    function void main () {
        var int n;
        let n = Keyboard.readInt("Enter n: ");
        do Output.printString("The result is: ");
        do Output.printInt(sum(n));
        return;
    }
    // More Main methods.
}
```

Example: Object-Oriented Programming

```java
/** Represents a bank account. 
 * A bank account has an owner, an id, and a balance. 
 * The id values start at 0 and increment by 1 each 
 * time a new account is created. */
class BankAccount {
    /** Constructs a new bank account with a 0 balance. */
    constructor BankAccount (String owner) {
        var int id;
        let id = id + 1;
        let nAccounts = nAccounts + 1;
        let this.owner = owner;
        let balance = 0;
    }
    // More BankAccount methods.
}
```

Example: Object-Oriented Programming

```java
class BankAccount {
    static int nAccounts;
    field int id;
    field String owner;
    field int balance;
    /** Hendels deposits */
    method void deposit (int amount) {
        let balance = balance + amount;
        return;
    }
    /** Hendels withdraws */
    method void withdraw (int amount) {
        let amount = amount > balance;
        let balance = balance - amount;
        return;
    }
    // More BankAccount methods.
}
```

Example: Object-Oriented Programming

```java
class BankAccount {
    static int nAccounts;
    field int id;
    field String owner;
    field int balance;
    /** Prints information about this account. */
    method void printInfo () {
        do Output.printInt(id);
        do Output.printString(owner);
        do Output.printInt(balance);
        return;
    }
    // More BankAccount methods.
}
```

Example: Object-Oriented Programming

```java
class BankAccount {
    static int nAccounts;
    field int id;
    field String owner;
    field int balance;
    /** Prints information about this account. */
    method void printInfo () {
        do Output.printInt(id);
        do Output.printString(owner);
        do Output.printInt(balance);
        return;
    }
    // More BankAccount methods.
}
```

Example: Object-Oriented Programming

```java
class BankAccount {
    static int nAccounts;
    field int id;
    field String owner;
    field int balance;
    /** Prints information about this account. */
    method void printInfo () {
        do Output.printInt(id);
        do Output.printString(owner);
        do Output.printInt(balance);
        return;
    }
    // More BankAccount methods.
}
```

Example: Procedural Programming

```java
class Main {
    /** Sums up 1 + 2 + 3 + ... + n */
    function int sum (int n) {
        var int sum = 1;
        let sum = sum + 1;
        while (<n > n) {
            let sum = sum + 1;
            let i = i + 1;
        }
        return sum;
    }
    function void main () {
        var int n;
        let n = Keyboard.readInt("Enter n: ");
        do Output.printString("The result is: ");
        do Output.printInt(sum(n));
        return;
    }
    // More Main methods.
}
```

Example: Object-Oriented Programming

```java
/** Represents a bank account. */
class BankAccount {
    // class-level variable 
    static int nAccounts;
    // Private variables (aka fields / properties)
    field int id;
    field String owner;
    field int balance;
    /** Constructs a new bank account */
    constructor BankAccount new (String owner) {
        let id = id + 1;
        let nAccounts = nAccounts + 1;
        let this.owner = owner;
        let balance = 0;
    }
    // More BankAccount methods.
}
```

Example: Object-Oriented Programming

```java
class BankAccount {
    static int nAccounts;
    field int id;
    field String owner;
    field int balance;
    /** Constructs a new bank account */
    constructor BankAccount new (String owner) {
        let id = id + 1;
        let nAccounts = nAccounts + 1;
        let this.owner = owner;
        let balance = 0;
    }
    // More BankAccount methods.
}
```

Example: Object-Oriented Programming

```java
class BankAccount {
    static int nAccounts;
    field int id;
    field String owner;
    field int balance;
    /** Prints information about this account. */
    method void printInfo () {
        do Output.printInt(id);
        do Output.printString(owner);
        do Output.printInt(balance);
        return;
    }
    // More BankAccount methods.
}
```

Example: Object-Oriented Programming

```java
class BankAccount {
    static int nAccounts;
    field int id;
    field String owner;
    field int balance;
    /** Prints information about this account. */
    method void printInfo () {
        do Output.printInt(id);
        do Output.printString(owner);
        do Output.printInt(balance);
        return;
    }
    // More BankAccount methods.
}
```

Example: Object-Oriented Programming

```java
class BankAccount {
    static int nAccounts;
    field int id;
    field String owner;
    field int balance;
    /** Prints information about this account. */
    method void printInfo () {
        do Output.printInt(id);
        do Output.printString(owner);
        do Output.printInt(balance);
        return;
    }
    // More BankAccount methods.
}
```

Example: Object-Oriented Programming

```java
class BankAccount {
    static int nAccounts;
    field int id;
    field String owner;
    field int balance;
    /** Prints information about this account. */
    method void printInfo () {
        do Output.printInt(id);
        do Output.printString(owner);
        do Output.printInt(balance);
        return;
    }
    // More BankAccount methods.
}
```

Example: Object-Oriented Programming

```java
class BankAccount {
    static int nAccounts;
    field int id;
    field String owner;
    field int balance;
    /** Prints information about this account. */
    method void printInfo () {
        do Output.printInt(id);
        do Output.printString(owner);
        do Output.printInt(balance);
        return;
    }
    // More BankAccount methods.
}
```

Example: Object-Oriented Programming

```java
class BankAccount {
    static int nAccounts;
    field int id;
    field String owner;
    field int balance;
    /** Prints information about this account. */
    method void printInfo () {
        do Output.printInt(id);
        do Output.printString(owner);
        do Output.printInt(balance);
        return;
    }
    // More BankAccount methods.
}
```

Example: Object-Oriented Programming

```java
class BankAccount {
    static int nAccounts;
    field int id;
    field String owner;
    field int balance;
    /** Prints information about this account. */
    method void printInfo () {
        do Output.printInt(id);
        do Output.printString(owner);
        do Output.printInt(balance);
        return;
    }
    // More BankAccount methods.
}
```

Example: Object-Oriented Programming

```java
class BankAccount {
    static int nAccounts;
    field int id;
    field String owner;
    field int balance;
    /** Prints information about this account. */
    method void printInfo () {
        do Output.printInt(id);
        do Output.printString(owner);
        do Output.printInt(balance);
        return;
    }
    // More BankAccount methods.
}
```

Example: Object-Oriented Programming

```java
class BankAccount {
    static int nAccounts;
    field int id;
    field String owner;
    field int balance;
    /** Prints information about this account. */
    method void printInfo () {
        do Output.printInt(id);
        do Output.printString(owner);
        do Output.printInt(balance);
        return;
    }
    // More BankAccount methods.
}
```

Example: Object-Oriented Programming

```java
class BankAccount {
    static int nAccounts;
    field int id;
    field String owner;
    field int balance;
    /** Prints information about this account. */
    method void printInfo () {
        do Output.printInt(id);
        do Output.printString(owner);
        do Output.printInt(balance);
        return;
    }
    // More BankAccount methods.
}
```

Example: Object-Oriented Programming

```java
class BankAccount {
    static int nAccounts;
    field int id;
    field String owner;
    field int balance;
    /** Prints information about this account. */
    method void printInfo () {
        do Output.printInt(id);
        do Output.printString(owner);
        do Output.printInt(balance);
        return;
    }
    // More BankAccount methods.
}
```

Example: Object-Oriented Programming

```java
class BankAccount {
    static int nAccounts;
    field int id;
    field String owner;
    field int balance;
    /** Prints information about this account. */
    method void printInfo () {
        do Output.printInt(id);
        do Output.printString(owner);
        do Output.printInt(balance);
        return;
    }
    // More BankAccount methods.
}
```

Example: Object-Oriented Programming

```java
class BankAccount {
    static int nAccounts;
    field int id;
    field String owner;
    field int balance;
    /** Prints information about this account. */
    method void printInfo () {
        do Output.printInt(id);
        do Output.printString(owner);
        do Output.printInt(balance);
        return;
    }
    // More BankAccount methods.
}
```

Example: Object-Oriented Programming

```java
class BankAccount {
    static int nAccounts;
    field int id;
    field String owner;
    field int balance;
    /** Prints information about this account. */
    method void printInfo () {
        do Output.printInt(id);
        do Output.printString(owner);
        do Output.printInt(balance);
        return;
    }
    // More BankAccount methods.
}
```
Jack Language Specification

- Syntax
- Data types
- Variable kinds
- Expressions
- Statements
- Subroutine calling
- Program structure
- Standard library

(for complete language specification, see the book).

Jack Data Types

- Primitive types:
  - int
  - char

- Abstract data types:
  - String
  - Array

- Application-specific types:
  - BankAccount
  - Fraction
  - List

Jack expressions

- A Jack expression is any one of the following:
  - A constant
  - A variable in scope (the variable may be static, field, local, or a parameter)
  - The keyword this, denoting the current object
  - An array element using the syntax arrayName[expression],
    where arrayName is a variable of type Array in scope
  - A subroutine call that returns a non-void type
  - An expression prefixed by one of the unary operators, e.g., ~ or -
    expression
  - An expression of the form expression op expression where op is one of the following:
    + - / % (integer arithmetic operators)
    & | (boolean and and or operators, bit-wise)
    < > (comparison operators)
    [ ] (an expression within parentheses)
**Jack Statements**

General syntax: `expression`;  
where each argument is a valid Jack expression

Parameter passing is by-value (primitive types) or by-reference (object types)

Example 1:

Consider the function (static method): `function int sqrt(int n)`

This function can be invoked (in the same class) as follows:

```
sqrt(17)
sqrt(x) = sqrt(b * b - (4 * a * c))
sqrt(a * sqrt(c - 17) + 3)
```

Etc. In all these examples the argument value is computed and passed by-value.

**Jack Subroutine Calls**

General syntax: `subroutineName(args0, args1, ...)`

where each argument is a valid Jack expression

Parameter passing is by-value (primitive types) or by-reference (object types)

Example 1:

Consider the method: `method Matrix plus(Matrix other)`

If `u` and `v` were variables of type `Matrix`, this method can be invoked using:

```
u.plus((v)
```

The `v` variable is passed by-reference, since it refers to an object.

**Noteworthy Features**

- The (cumbersome) `let` keyword, as in `let x = 0;`
- The (cumbersome) `do` keyword, as in `do reduce();`
- No operator priority:
  - `1 + 2 * 3` yields 9, since expressions are evaluated left-to-right;
  - To effect the commonly expected result, use `1 + (2 * 3)`
- Only three primitive data types: int, boolean, char;
- In fact, each one of them is treated as a 16-bit value
- No casting; a value of any type can be assigned to a variable of any type
- Array declaration: `Array x;` followed by `x = Array.new(sz);`
  - Arrays can hold any type.
- Static methods are called `Function`
- Constructor methods are called `constructor`;
- Invoking a constructor: `ClassName.constructorName(argsList)`

**Class Structure**

- Field variable declarations;  
  static variable declarations;  
  constructor (parameterList) {  
  local variable declarations;  
  statements  
  }
- method (parameterList) {  
  local variable declarations;  
  statements  
  }
- function (parameterList) {  
  local variable declarations;  
  statements  
  }

**About this spec:**

- Every part in this spec can appear 0 or more times
- The order of the field / static declarations is arbitrary
- The order of the subroutine declarations is arbitrary
- Each type is either int, boolean, char, or a class name.

**Jack Standard Library aka Jack OS**