INFO 5100

Application Engineering Design

Java Arrays and Loops

Daniel Peters

d.peters@neu.edu

- Lecture
  1. Array
  2. Range Loop
  3. Iterator Loop
  4. ListIterator Loop

Linear Algebra: Scalar Data Type

- Single data type: int, double, char, float
- One data value stored in one location

- Examples:

int integer = 1;

double n = 3.9;

char c = 'A';

String name = "Dan";

Linear Algebra: Vector Data Type

- Multiple Scalar data types
- Multiple elements of the same type

- Examples:

int[] integers = { 1,2,3,4,5,6,7 };

double[] numbers = { 3.9, 4.0, 4.1 };

char[] characters = {'A', 'B', 'C', 'D' };

String[] names = { "Adam", "Eve" };

Array

- Contains multiple items of same type
- Items stored in contiguous memory
- Fixed size
- Mutable data type
  - String data type is immutable and interned
- Supports Random access
  - Access any element in constant time
- Very efficient
  - Smallest memory footprint for data storage

Array

Syntax:

 String[] names = {"adam", "eve"};

 int[] numbers = { 1, 2, 3 };

 char[] characters = { 'A','B','C' };

 String[] strArray = new String[3];

- Supports for primitive and reference types
- Array of characters is NOT a String

Array Example 1

int[] **numbers** = { 1,2,3 };   // **array of int types**

System.*out.print(*
        numbers*[0] +", "*
    + **numbers***[1] + ", "*
    + **numbers***[2] + ", ");*

Produces Output:

1, 2, 3,

Array Example 2

String[] names = { "Peter", "Paul", "Mary"};

names[1] = "Noel Paul Stookey";

System.*out.print(*

     names[0] +*", "*

     + names[1] + *", "*

     + names[2] + *", ");*

Produces Output:

Peter, Noel Paul Stookey, Mary,

Array Example 3

int[] **numbers** = { 1,2,3,4 };   // **array of int types**

System.*out.print(***numbers**.*length* +": ");

for (int n : **numbers**) {

      System.*out.print(n +", "); // 1,2,3,4,*

} // range based for loop

Produces Output:

4: 1, 2, 3, 4,

Array Example 4

```
int[] numbers = new int[7];        // array of int types
for (int i=0; i < 7; i++) {
        numbers[i] = i + 1;
} // i is available ONLY for the loop, not rest of code
for (int number : numbers ) {
        System.out.print(number +", ");
}
```

Produces Output:

1, 2, 3, 4, 5, 6, 7,

Array Example 4

```
int[] numbers = new int[7];       // array of int types
int i=0;
for (  ; i < 7; ) { numbers[i] = i + 1; i++; }
for (int number : numbers ) {
        System.out.print(number +", ");
}
```

Produces Output:

1, 2, 3, 4, 5, 6, 7,

Array Example 4

```
int[] numbers = new int[7];      // array of int types
int i=0;
for (  ; i < 7; i++) { numbers[i] = i + 1; }
for (int number : numbers ) {
        System.out.print(number +", ");
} // variable i remains available for code after loop
```

Produces Output:

1, 2, 3, 4, 5, 6, 7,

Array Example 4

```
int[] numbers = new int[7];        // array of int types
for (int i=0, int j = 1; i < 7; i++,j++) {
        numbers[i] = i + j;
} // init both i and j as variables
for (int number : numbers ) {
        System.out.print(number +", ");
}
```

Produces Output:

1, 2, 3, 4, 5, 6, 7,

Range Based Loop

Syntax:

```
for ( type item : container) { // loop body   }
```

Example:

```
String[] names =
    {"Adam", "Eve"};
for (String name : names) {
    System.out.println(name);
}
```

Array Examples

String[] **names** = {"Adam", "Eve"}; // String array

for (String name : **names**) {

       System.out.print(name +", ");

}

Produces Output:

Adam, Eve,

Array Examples

String[] threeFruit = {"Apple", "Pear", "Orange"};

for (String fruit : threeFruit) {
       System.***out.print(fruit +", ");***
}

- Produces Output:

Apple, Pear, Orange,

ArrayList

- Sequential container class like array
- Contains multiple items of same type
- Items stored in contiguous memory
- Unlike Array, NOT Fixed size
  - Can grow and shrink
- Supports Random access
- Requires more storage than an array for same number of elements
- REFERENCE TYPES (objects) ONLY

ArrayList

Syntax:

ArrayList<RefType> listName;

listName = new ArrayList<RefType>();

RefType e1 = new RefType();

listName.add(e1);

RefType e2 = new RefType();

listName.add(e2);

Java Collections:

- Sequential Containers Interface
  - **List**
- Sequential Containers Classes
  - **ArrayList**
  - **Vector**
  - **LinkedList**
- Similar in use
- Different implementations and benefits
- Each implements List interface

ArrayList Example

ArrayList<String> names;

names = **new ArrayList<String>();**

names.add("Peter");

names.add("Paul");

names.add("Mary");

Range Loop Example

List<String> **names** = new **ArrayList**<String>();

names.add("Peter");

names.add("Paul");

names.add("Mary");

for (String *name* : **names**) {

      System.out.print(*name* +", ");

}

Produces Output:

Peter, Paul, Mary

Range Loop Example

List<String> names = new ArrayList**<>** ();

- NOTE:
  - Parameterized type for ArrayList container
    - <String> can be explicitly declared
    - *<> can be inferred* by compiler because of assignment (=) to **List<String>** names;

Range Loop Example

List<String> names = **new ArrayList**<>(
Arrays.*asList*("Peter", "Paul", "Mary"));

for (String *name* : **names**) {

System.out.print(*name* +", ");

}

Produces Output:

Peter, Paul, Mary

Iterator Loop

Syntax:

    Iterator<String> it;   // iterator for type String

    Example:

    List< String > **names** = new ArrayList<>();

    Iterator< String > it = **names**.iterator();

- Use with collections supporting Iterator interface

Iterator Loop

List<String> names = new Vector<String>();

Iterator<String> **it** = names.iterator()

while (**it**.hasNext()) {

   System.out.println(**it**.next());

 }

- Loop until all elements in list are accessed

Iterator Loop Example

```java
List<Integer> numbers = new ArrayList<Integer>();
numbers.add(1);
numbers.add(2);
numbers.add(3);

Iterator<Integer> it = numbers.iterator();
while (it.hasNext()) {
        System.out.print(it.next() +", ");
}
```

Produces Output:

1, 2, 3,

Iterator Loop Example

List<String> names= new ArrayList<String>();

names.add("Peter");

names.add("Paul");

names.add("Mary");

Iterator<String> it = names.iterator();

while (it.hasNext()) {

       System.*out.print(it.next() +", ");*

}

Produces Output:

Peter, Paul, Mary,

ListIterator Loop

Syntax:

ListIterator<Integer> it;

Example:

ListIterator<Integer> it = integerList.listIterator();

- Use with collections supporting ListIterator interface
  - **ArrayList**, **Vector**, **LinkedList**
- Can iterate *forward* and *backwards*
- Can *modify* element (**set**)

ListIterator Loop

List<Integer> numbers = **new** ArrayList<>(
Arrays.*asList*(1, 2, 3));

ListIterator <Integers> *it* = numbers.**listIterator**()

while (*it*.**hasNext()**) {

System.out.println(*it*.**next()** + ", ");
}

while (*it*.**hasPrevious()**) {

System.out.println(*it*.**previous()** + ", ");
}

Produces Output:

1, 2, 3, 3, 2, 1,

ListIterator Loop Example

List<String> names= **new ArrayList<String>();**

names.add("Peter");

names.add("Paul");

names.add("Mary");

ListIterator<String> it = names.**listIterator**();

while (it.hasNext()) {

        String element = it.**next**();

        element = element.toUpperCase() + ", ";

        it.**set**(element); // ONLY in *ListIterator*

        System.***out.print(***element +***", ");   // changed element***

}

Produces Output:

PETER, PAUL, MARY

ListIterator Loop Example

List< Integer > numbers= **new ArrayList< >();**

numbers.add(1);

numbers.add(2);

numbers.add(3);

ListIterator<Integer> it = numbers.listIterator();

while (it.hasNext()) {

      System.out.print*(it.next() +", ");*

}

Produces Output:

1,2,3,