RMIT Classification: Trusted

INTE2512 Object-Oriented Programming

Strings & Arrays

Quang Tran

RMIT UNIVERSITY

Outline

- Strings
- Regular Expressions
- Arrays
- Multidimensional Arrays
- Arrays Class
- ArrayList
- LinkedList
- HashMap
- HashSet

Strings

- A string is simply an array of characters
- Java provides the String class (**object** data type) and several methods to allow working with strings conveniently
- Create strings

```
String greeting = new String("Hello");
String greeting = "Hello";
```

- String length

```
System.out.println(greeting.length());
```

- String comparison

```
System.out.println(greeting.equals("hello"));
```

Strings

- String concatenation

```
String firstName = "John"; String lastName  = "Doe";
String fullName  = firstName + " " + lastName;
System.out.println(fullName);
```

- Substring

```
String txt = "RMIT University Vietnam";
System.out.println(txt.substring(5, 8));
```

- Searching in a string

```
System.out.println(txt.indexOf("Viet"));
System.out.println(txt.indexOf("Hello"));
```

Strings

Method	Description
int length()	Returns the number of character in this string
boolean isEmpty()	Returns true if, and only if, length() is 0
boolean equals(String str)	Returns true if this string is equal to str
boolean equalsIgnoreCase(String str)	Returns true if this string is equal to str, ignoring case differences
int compareTo(String str)	Compares two strings lexicographically
int compareToIgnoreCase(String str)	Compares two strings lexicographically, ignore case differences
char charAt(int index)	Returns the character at the given index
int codePointAt(int index)	Returns the Unicode code point of the character at the given index
String concat(String str)	Returns a new string that concatenates this string with str
String toUpperCase()	Returns a new string with all letters in uppercase
String toLowerCase()	Returns a new string with all letters in lowercase
String substring(int beginIndex, int endIndex)	Returns a new string that is a substring of this string

Strings

Method	Description
boolean startsWith(String str)	Returns true if this string starts with str
boolean endsWith(String str)	Returns true if this string ends with str
boolean contains(String str)	Returns true if this string contains str
int indexOf(String str)	Returns the index of the first occurrence of str in this string , or -1 if there is no such occurrence
int lastIndexOf(String str)	Returns the index of the last occurrence of str in this string , or -1 if there is no such occurrence
String replace(char oldChar, char newChar)	Returns a new string resulting from replacing all occurrences of oldChar in this string by newChar
String replaceAll(String regex, String replacement)	Replaces each substring in this string that matches the given regex with the given replacement
String[] split(String regex)	Splits this string around matches of the given regex
boolean matches(String regex)	Returns true if this string matches the given regex

Strings

- Conversion between strings and numbers

```
String intString = "123";
int intValue = Integer.parseInt(intString);
String doubleString = "3.14159";
double doubleValue = Double.parseDouble(doubleString);
```

Strings

- In Java, strings are **immutable** – they can’t be changed after created
- The following code creates a new string and assigns it back to the original variable:

```
String greeting = "Hello";

greeting += " World";
```

- More details can be found in the String API

Regular Expressions

- A regular expression (regex) is a string that describes a pattern in a sequence of characters
- Regex can be used for matching, replacing, and splitting strings

```
String s1 = "Java is fun";
String s2 = "Java is cool";
String regex = "Java.*";
System.out.println(s1.matches(regex));
System.out.println(s2.matches(regex));
```

Regular Expressions

Regular expression	Matches
x	a specified character x
.	any single character
(ab|cd)	ab or cd
[abc]	a, b, or c
[^abc]	any character except a, b, or c
[a-z]	any character from a through z
[^a-z]	any character except a through z
[a-e[m-p]]	a through e or m through p
[a-e&&[c-p]]	intersection of a-e with c-p
\d	a digit, same as [0-9]
\D	a non-digit

Regular Expression

Regular expression	Matches
\w	a word character, same as [a-zA-Z0-9_]
\W	a non-word character, same as [^a-zA-Z0-9_]
\s	a whitespace character
\S	a non-whitespace character
p*	zero or more occurrences of pattern p
p+	one or more occurrences of pattern p
p?	zero or one occurrence of pattern p
p{n}	exactly n occurrences of pattern p
p{n, }	at least n occurrences of pattern p
p{n, m}	between n and m occurrences of pattern p (inclusive)
^	beginning of a line
$	end of a line

Quiz

- What is the regular expression of an even integer?

  Answer: (\d)*[02468]

- The Social Security Number (SSN) in the US has the format xxx-xx-xxxx, where x is a digit. What is the regular expression for SSN?

  Answer: \d{3}-\d{2}-\d{4}

Arrays

- An array is a **fixed-size order** collection of items of the same data type
- Create arrays

```
int[] nums;                 // declare an array
nums = new int[4];          // specify array length(size)
String[] cars = new String[4]; // combine in one statement
```

- Create and initialize arrays

```
int[] nums = {5, 10, 15, 20};
String[] cars = {"Honda", "Toyota", "Ford", "BMW"};
```

Arrays

- Access and update an item

```
String[] cars = {"Honda", "Toyota", "Ford", "BMW"};
System.out.println(cars[2]);
cars[2] = "Kia";
System.out.println(cars[2]);
```

- Loop through an array using a for statement and array indexes

```
for (int i = 0; i < cars.length; i++) {
    System.out.println(cars[i]);
}
```

Arrays

- Loop through an array using a for-each statement

```
String[] cars = {"Honda", "Toyota", "Ford", "BMW"};
for (String car : cars) {
    System.out.println(car);
}
```

Multidimensional Arrays

- Create a multidimensional array

```
int[][] chessCells = new int[8][8];
```

- Create and initialize a multidimensional array

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

- Access multidimensional array items

```
for (int i = 0; i < myNumbers.length; i++) {
    for (int j = 0; j < myNumbers[i].length; j++) {
        System.out.print(myNumbers[i][j] + " ");
    }
    System.out.println();
}
```

Arrays Class

- Arrays is a utility class in the java.util package
- It contains several static methods for copying, sorting, searching, comparing arrays, and filling array elements
- These methods are **overloaded** (methods with the same name but different parameters) for all primitive data types
- The next slide shows the common methods of the Arrays class for int

Arrays Class

Method	Description
static int binarySearch(int[] a, int key)	Searches and returns the index of the key, if found, in the specified array of ints using the binary search algorithm
static int copyOf(int[] original, int newLength)	Copies the specified array, truncating or padding with zeros (if necessary) so the copy has the specified length
static int copyOfRange(int[] original, int from, int to)	Copies the specified range of the specified array into a new array
static boolean equals(int[] a, int[] a2)	Returns true if the two specified arrays of ints are equal
static int compare(int[] a, int[] a2)	Compares two int arrays lexicographically
static void fill(int[] a, int val)	Assigns the specified int value to each element of the specified array of ints
static void sort(int[] a)	Sorts the specified array into ascending numerical order

ArrayList

- ArrayList is a **resizable** array class in the java.util package
- Create an ArrayList object then add items to it

```
import java.util.ArrayList;
public class Main {
    public static void main(String[] args) {
        ArrayList<String> cars = new ArrayList<String>();
        cars.add("Audi");
        cars.add("BMW");
        cars.add("Ford");
        cars.add("Mazda");
        System.out.println(cars);
    }
}
```

ArrayList

- Get, set, and remove an item with an index

```
import java.util.ArrayList;
public class Main {
    public static void main(String[] args) {
        ArrayList<String> cars = new ArrayList<String>();
        cars.add("Audi");
        cars.add("BMW");
        cars.add("Ford");
        cars.add("Mazda");
        cars.set(3, "Toyota");
        cars.remove(1);                                     // remove one element
        for (int i = 0; i < cars.size(); i++) {
            System.out.println(cars.get(i));
        }
        cars.clear();                                       // remove all elements
    }
}
```

ArrayList

- Sort an ArrayList object then loop through its elements

```
import java.util.ArrayList;
import java.util.Collections;
public class Main {
    public static void main(String[] args) {
        ArrayList<String> cars = new ArrayList<String>();
        cars.add("Volvo");
        cars.add("BMW");
        cars.add("Ford");
        cars.add("Mazda");
        Collections.sort(cars);
        for (String car : cars) {
            System.out.println(car);
        }
    }
}
```

LinkedList

- The LinkedList class has all the methods in the ArrayList class

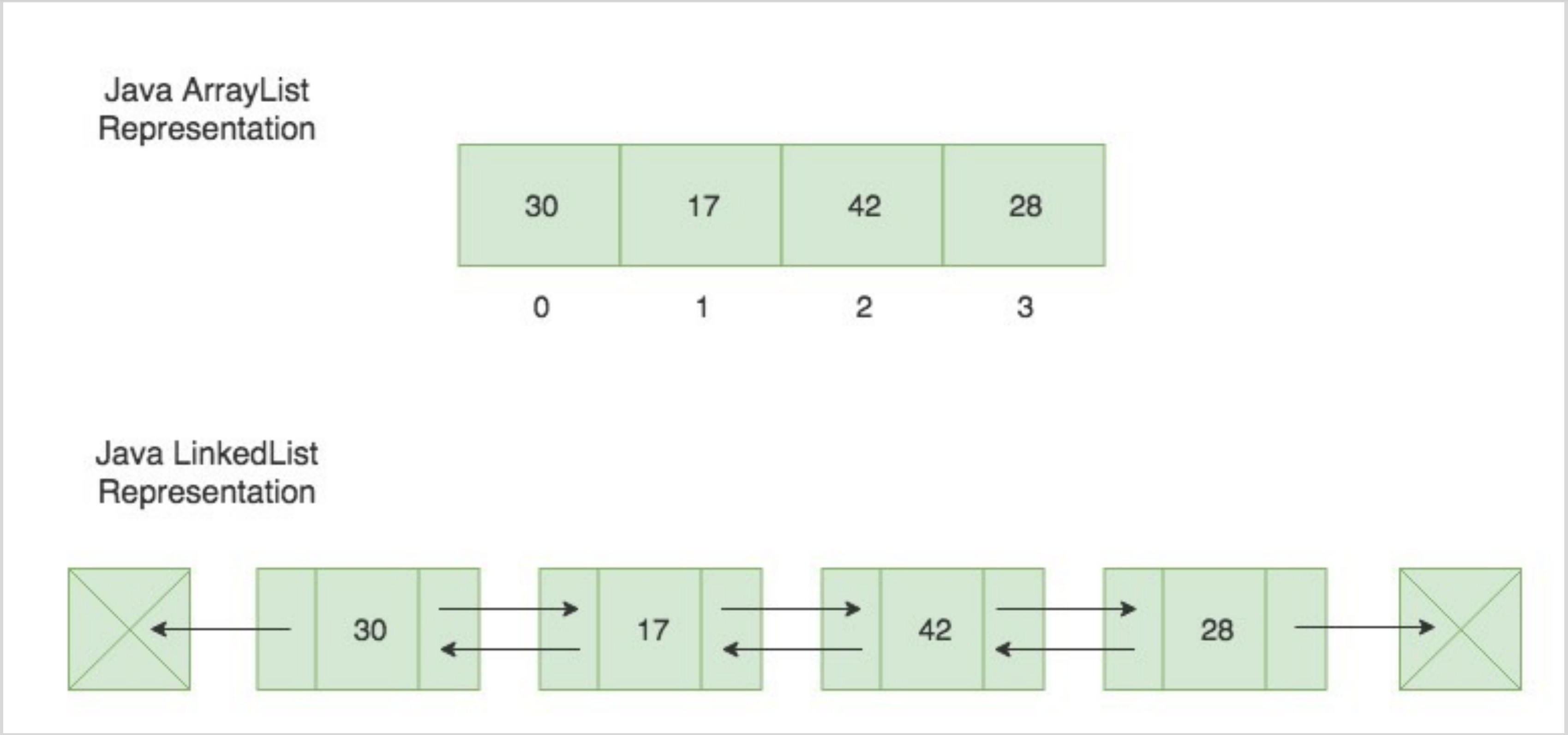
```
import java.util.LinkedList;
import java.util.Collections;
public class Main {
    public static void main(String[] args) {
        LinkedList <String> cars = new LinkedList<String>();
        cars.add("Volvo");
        cars.add("BMW");
        cars.add("Ford");
        cars.add("Mazda");
        Collections.sort(cars);
        for (String car : cars) {
            System.out.println(car);
        }
    }
}
```

LinkedList

- But it also has some additional methods to do a certain operations more efficiently

Method	Description
addFirst()	Adds an element to the beginning of the list
addLast()	Add an element to the end of the list
removeFirst()	Remove an element from the beginning of the list
removeLast()	Remove an element from the end of the list
getFirst()	Get the element at the beginning of the list
getLast()	Get the element at the end of the list

ArrayList vs LinkedList



ArrayList vs LinkedList

- ArrayList is used more often, but there are scenarios where LinkedList is preferred

ArrayList	LinkedList
Implemented as an array. If the array is not big enough, a larger one is created to replace the old one.	Implemented as a doubly-linked list. Can grow and shrink at will.
Accessing an item is very fast	Accessing an item is slow in average
Adding and removing items are slow	Adding and removing items are fast

HashMap

- A HashMap object stores an **unorder** collection of items in "key-value" pairs of object data types
- Create a HashMap object then add items to it

```
import java.util.HashMap;
public class Main {
    public static void main(String[] args) {
        HashMap<String, String> capitalCities = new HashMap<String, String>();
        capitalCities.put("England", "London");
        capitalCities.put("Germany", "Berlin");
        capitalCities.put("USA", "Washington DC");
        System.out.println(capitalCities);
    }
}
```

HashMap

- Each item can be accessed or removed by the "key"

```
import java.util.HashMap;
public class Main {
    public static void main(String[] args) {
        HashMap<String, String> capitalCities = new HashMap<String, String>();
        capitalCities.put("England", "London");
        capitalCities.put("Germany", "Berlin");
        capitalCities.put("USA", "Washington DC");
        System.out.println("Capital: " + capitalCities.get("Germany"));
        capitalCities.remove("Germany");
        System.out.println("Capital: " + capitalCities.get("Germany"));
        System.out.println("size: " + capitalCities.size());
        capitalCities.clear();
        System.out.println("size: " + capitalCities.size());
    }
}
```

HashMap

- Loop through a HashMap object

```
import java.util.HashMap;
public class Main {
    public static void main(String[] args) {
        HashMap<String, String> capitalCities = new HashMap<String, String>();
        capitalCities.put("England", "London");
        capitalCities.put("Germany", "Berlin");
        capitalCities.put("USA", "Washington DC");
        for (String country : capitalCities.keySet()) {
            System.out.print("key: " + country);
            System.out.println(", value: " + capitalCities.get(country));
        }
    }
}
```

HashSet

- A HashSet object stores an **unorder** collection of items of object data types where every item is unique
- Create a HashSet object then add items to it

```
import java.util.HashSet;
public class Main {
    public static void main(String[] args) {
        HashSet<String> cars = new HashSet<String>();
        cars.add("Audi");
        cars.add("BMW");
        cars.add("Ford");
        cars.add("BMW");
        System.out.println(cars);
    }
}
```

HashSet

- Check and remove items

```
import java.util.HashSet;
public class Main {
    public static void main(String[] args) {
        HashSet<String> cars = new HashSet<String>();
        cars.add("Audi");
        cars.add("BMW");
        cars.add("Ford");
        System.out.println(cars);
        System.out.println("Contains Ford: " + cars.contains("Ford"));
        cars.remove("Audi");
        System.out.println(cars);
        cars.clear();
        System.out.println(cars);
    }
}
```

HashSet

- Check and remove items

```
import java.util.HashSet;
public class Main {
    public static void main(String[] args) {
        HashSet<String> cars = new HashSet<String>();
        cars.add("Audi");
        cars.add("BMW");
        cars.add("Ford");
        System.out.println(cars);
        System.out.println("Contains Ford: " + cars.contains("Ford"));
        cars.remove("Audi");
        System.out.println(cars);
        cars.clear();
        System.out.println(cars);
    }
}
```

HashSet

- Loop through a HashSet object

```
import java.util.HashSet;
public class Main {
    public static void main(String[] args) {
        HashSet<String> cars = new HashSet<String>();
        cars.add("Audi");
        cars.add("BMW");
        cars.add("Ford");
        for (String car : cars) {
            System.out.println(car);
        }
        System.out.println("There are " + cars.size() + " cars");
    }
}
```

References

1. D. Y. Liang, Intro to Java Programming, 10th edition, chapter 1-5, 2015.
2. W3Schools, Java Tutorial, 2021.
3. TutorialsPoint, Java Tutorial, 2021.
4. Jenkov, Java Tutorial, page 1-20, 2021.
5. Oracle Corporation, Java 11 API Specification, 2019.