Chapter 6: Data Abstraction

• Java is an object-oriented programming language
• In this chapter, we will see how to define, create, and use objects in our programs
• They are called objects because they are often used to represent 'real' objects such as employees, students, or planets, or airplane flights, or ...
  – nearly anything that is too complex to represent with a primitive type or an array

In Java, there are three types of data values
– primitives
– arrays
– objects
  • actually, arrays are a special type of object

Class

• In Java, objects are used to represent data values
• Just as 3 is a primitive value of type int, every object must have a type
  – These types are called classes

A class defines a type of object, including
– its data: the information that is used to represent the object
– the operations (methods) that can be performed on the data
• After the type is defined, objects of that type can be defined and used
• An individual value of a class is called an object, or an instance of the class
Example

- Suppose we want to represent planets in Java
- We can define a class `Planet`, with
  - data: diameter, mass, distance from the sun, orbital period, rotational period, location, ...
  - methods:
    - `setDiameter`, `setMass`, etc.
    - `getDiameter`, `getMass`, etc.
    - `move`
- Instances of `Planet` might include earth, mars, venus, saturn

Objects and Methods

- Different objects have different methods for manipulating their data
- The specific methods are determined based on what makes sense for that type of object
- For example, with strings:
  - `length`, `concatenation`, `comparison`, `substring`

String: A standard class

- String is a standard Java class
- Remember, values from a class are called objects, so
  - "hello" is an object from the class `String`, or an instance of the class
- The class `String` has instance methods that operate on an instance of the class
- For example:
  - `length()`, `concat()`, `compareTo()`, `charAt()`

Example: Palindromes

- Palindrome: A word that reads the same forwards and backwards
  - "eye", "madam", "radar"
- Two String methods
  - `length()` - returns the length of the String
  - `charAt()` - returns the character at a given position in the String
  - numbered from 0 to `length() - 1`
Algorithm

• Get a word from the user
• Compare the first and last characters
  – If they are different, return false
  – Otherwise, repeat with the second and second to last characters, etc.
• If the characters all match, return true

Revised Algorithm

• Set left to the index of the first (leftmost) character
• Set right to index the last (rightmost) character
• While left is less than right
  – Compare the left character with the right character
  – If they are not equal return false
  – Increment left
  – Decrement right
• Return true

// Palindrome.java - check if a string is a palindrome
public class Palindrome {
    public static void main(String[] args) {
        System.out.print("Please enter a string: ");
        String str = Console.in.readWord();
        System.out.println(str + " " + isPalindrome(str));
    }
    static boolean isPalindrome(String s) {
        int left = 0;
        int right = s.length() - 1;
        while (left < right) {
            if (s.charAt(left) != s.charAt(right))
                return false;
            left++;
            right--;
        }
        return true;
    }
}

Methods

• Each type of object supports a specified set of methods
• There are two types of methods in a class
  – instance methods
  – class methods
Instance Methods

• Instance methods are called for a specific object and have direct access to that object’s data without having to pass the object as a parameter

  String s = "abc";
  s.length();

• The object is implicitly passed as a parameter to the method.
• We use the object's name followed by a '.', followed by the method name

String Instance Methods

• boolean equals( Object anObject )
  – compares this String with another object
  – not the same as ‘==’

• int length()
  – Number of characters in this String

• char charAt( int index )
  – The character at position index in this String
  – index ranges from 0 to length() - 1
  – IndexOutOfBoundsException

String Instance Methods 2

• int compareTo(String str)
  – Returns an integer value, based on lexicographic order

• int indexOf(int ch)
  – Index of where the first occurrence of ch occurs in this string or -1 if not present

• int indexOf(String str)
  – Index of the first character of a matching substring str

String Instance Methods 3

• String concat(String str)
  – Concatenates this string instance with str and returns the result

• String toLowerCase()
  – Returns a copy of this string, but converted to all lower case

• String toUpperCase()
  – Returns a copy of this string, but converted to all upper case
Class Methods

- These methods are part of the class definition, but don't operate on a specific instance of the class.
- Class methods are indicated by the keyword `static`.
  - We have written class methods up to now.
- Class methods provide some type of operation related to the objects provided by the class.
  - `Math.random()`
  - `Math.sqrt()`

String.valueOf()

- `static String valueOf( type prim )`
  - Returns the String representation of the value of `prim`.
  - `valueOf` is an overloaded function:
    - `valueOf( char c )`
    - `valueOf( char[] data )`
    - `valueOf( int i )`
    - `valueOf( double d )`

StringTest.java

```java
// StringTest.java - demo some String methods
public class StringTest {
    public static void main(String[] args) {
        String str1 = "aBcD", str2 = "abcd", str3;
        System.out.println(str1.equals(str2));
        System.out.println(str1.length());
        System.out.println(str1.charAt(1));
        System.out.println(str1.compareTo("aBcE"));
        System.out.println(str1.compareTo("aBcC"));
        System.out.println(str1.compareTo("aBcD"));
        System.out.println(str1.indexOf('D'));
        System.out.println(str1.indexOf("Bc"));
        System.out.println(str1.indexOf("zz"));
        // parts cut out here
    }
}
```
StringTest.java

```java
// StringTest.java - demo some String methods
public class StringTest {
    public static void main(String[] args) {
        String str1 = "aBcD", str2 = "abcd", str3;
        // parts cut out here
        System.out.println(str1.concat("efg"));
        str3 = str1.toLowerCase();
        System.out.println(str3);
        str3 = str1.toUpperCase();
        System.out.println(str3);
        System.out.println(str1);
        str3 = String.valueOf(123);
        System.out.println(str3.equals("123"));
    }
}
```

String Literals

• String literals are instances of String
• Therefore, we can invoke instance methods on them
  
  "abc".length();
  "hello".charAt(2);

String is unusual

• Because strings are so common, Java provides some special syntax for the class String
  — special support for concatenation
  — special support for String literals
• String is the only class that is treated specially

String Concatenation

• The '+' operator is overloaded to support String concatenation

  "hello " + "world"

  is equivalent to

  "hello ".concat("world");

• Instance methods are operations on objects
String Literals

- String literals are supported
  
  ```java
  String s = "hello";
  ```

  is equivalent to
  
  ```java
  char[] temp={'h','e','l','l','o'};
  String s = new String(temp);
  ```