Objects and Classes

Part 2 – Using Predefined Classes

Chapter 4, Core Java, Volume I

Contents

- Using Predefined Classes
- Object Reference Variables
- Working with LocalDate
- Accessors and Mutators
- Example: Displaying a Calendar

Using Predefined Classes in Java

- Consider the **Math** class (java.lang.Math, ref. Chapter 3)
  - Only contains a set of static methods; it doesn't need data.
  - You don't create objects to call **Math.sqrt(x)**.

- Consider the predefined **StringBuilder** class (java.lang.StringBuilder)
  - Alternative to **String** class (immutable strings)
  - Represents a mutable sequence of characters
  - Appending and inserting characters (building a string dynamically)
  - Methods
    - int length()
    - StringBuilder append(char ch)
    - StringBuilder append(String str)
    - StringBuilder insert(int offset, String str)
    - StringBuilder delete(int startIndex, int endIndex)
    - String toString()
    - …

Using Predefined Classes in Java

- Library Packages
  - java.lang (Fundamental classes; `Math`, `String`, etc.,) – you don't need to import it!
  - java.util (Utilit classes; `Scanner`, etc.)
  - java.swing (Swing GUI classes; `JFrame`, `JButton`, etc.)
  - java.io
  - java.net
  - …

Using Predefined Classes in Java

- Creating(instantiating) a StringBuilder Object
  - Using new operator
  - Calling constructor to initialize objects (a special method with the same name as the class)

```
StringBuilder sb = new StringBuilder();
```

- Calling (invoking) a method on an object
  - General structure

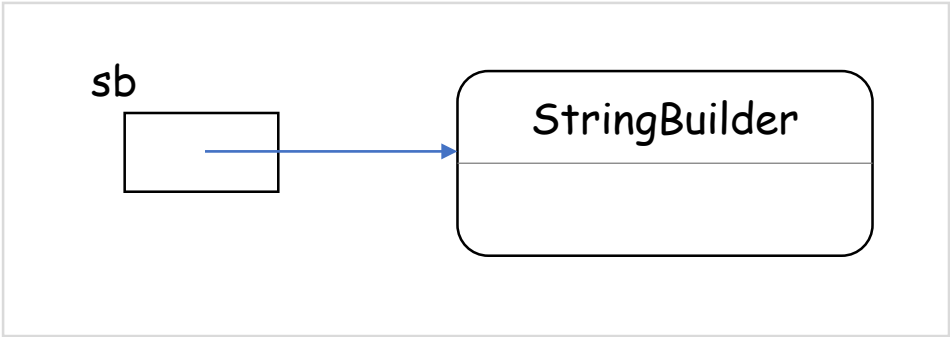
object.method_call

```
sb.append("Hello");          // append a string
sb.append('!');              // append a character
String s = sb.toString();    // return a String object with the same data, here "Hello!"
```

Object Reference Variables

- An object reference variable holds a *reference* to an object.

```
StringBuilder sb = new StringBuilder();
or
StringBuilder sb;
sb = new StringBuilder();
```



- A null reference or uninitialized reference

```
StringBuilder sb;       // sb refers to null (instance variables)
                        // sb is uninitialized (local variables)
```

  - Caution: Don't call a method on null reference or uninitialized variable

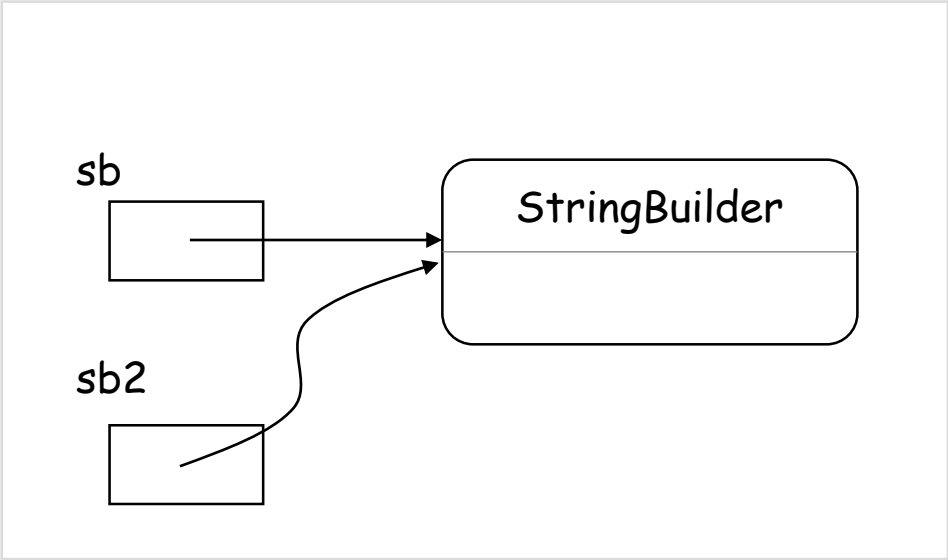
```
s = sb.toString();  // not yet
```

    - null : runtime error
    - uninitialized : compile-time error

Object Reference Variables

- Copying a variable makes a copy of the reference:

```
StringBuilder sb2 = sb;
```



Example: Working with LocalDate

- A LocalDate(java.time.LocalDate) is a date (day, month, year) in a particular location.

- Factory methods to create instances:

```
LocalDate rightNow = LocalDate.now();                    // static method
LocalDate newYearsEve = LocalDate.of(1999, 12, 31);
```

- LocalDate methods:

```
LocalDate aThousandDaysLater = newYearsEve.plusDays(1000);
year = aThousandDaysLater.getYear(); // 2002
month = aThousandDaysLater.getMonthValue(); // 09
day = aThousandDaysLater.getDayOfMonth(); // 26
```

- How is a LocalDate stored? How do these methods do their job? We don't know, and we don't care. That's encapsulation.

Accessor and Mutator Methods

- Accessor method doesn't modify object state.
  - All LocalDate methods are accessors. (immutable)

```
LocalDate aThousandDaysLater = newYearsEve.plusDays(1000);
year = aThousandDaysLater.getYear();
```

- Mutator methods modifies object state.
  - Older version of calendar date class has mutator methods:

```
GregorianCalendar someDay = new GregorianCalendar(1999, 11, 31);
someDay.add(Calendar.DAY_OF_MONTH, 1000);
  // someDay has been mutated
int year = someDay.get(Calendar.YEAR); // 2002

GregorianCalendar newYearsEve = new GregorianCalendar(1999, 12, 31);
newYearsEve.add(Calendar.DAY_OF_MONTH, 1);  // oops!!!
```

Example: Displaying a Calendar

```
import java.time.*;
public class CalendarTest
{
   public static void main(String[] args)
   {
      LocalDate date = LocalDate.now();
      int month = date.getMonthValue();
      int today = date.getDayOfMonth();

      date = date.minusDays(today - 1);
                  // Set to start of month
      DayOfWeek weekday = date.getDayOfWeek();
      int value = weekday.getValue();
                  // 1 = Monday, ... 7 = Sunday

     System.out.println("Mon Tue Wed Thu Fri Sat Sun");
     for (int i = 1; i < value; i++)
        System.out.print("    ");
      while (date.getMonthValue() == month)
      {
         System.out.printf("%3d", date.getDayOfMonth());
         if (date.getDayOfMonth() == today)
            System.out.print("*");
         else
            System.out.print(" ");
         date = date.plusDays(1);  // increment one day
         if (date.getDayOfWeek().getValue() == 1)
             System.out.println();  // Monday starts newline
      }
      if (date.getDayOfWeek().getValue() != 1)
            System.out.println();
   } // end of main
} // end of class
```