Strings

Topics

- Introduction
- Strings
- Concatenating strings
- Substrings
- Scanning strings
- Trimming strings
- Translating strings
- Checking strings
- StringBuffer and StringTokenizer

Unit objectives

After completing this unit, you should be able to:

- Represent a character string in Java
- Perform operations on the string, such as concatenation, substring extraction and location, and space trimming
- Create comparable Java capability from common RPG built-in functions and operations that manipulate strings

Introduction

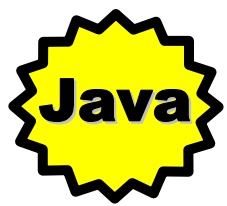
- RPG strings are multilen char fields.
- But Java char variables are length 1 only!
  - Java strings are instances of class String, not char variables.
- Why does Java use a class for strings?
  - Built-in data types are restricted to a few operators built into the language.
  - In RPG, additional function to strings is added with new string op-codes, and new string built-in functions.
  - In object-oriented languages, such as Java, much function is added with methods.
- In RPG, you define fixed length character field or array of characters as follows:

```
     Dmystring          S              40A   INZ('AnnaLisa')
     Dmystring2         S              10A   DIM(20)
```

String basics

- Strings are *objects* in Java, of the class String (in java.lang package, which is always implicitly imported for you).
- Although String is a class, the language also has some special built-in support for them:
  - You can concatenate with the "+" operator.
  - You do not have to use the **new** operator.

```
String text1 = new String("George");
String text2 = new String("Phil");
String finalText = new String(text1);
finalText = finalText.concat(" and ");
finalText = finalText.concat(text2);
```



OR

```
String text1 =  "George";
String text2 =  "Phil";
String finalText = text1 + " and " + text2;
```

Watch out! Gotchas

- String passed to concat appended to object.
  - Actual string target object is not affected.
  - Instead, a new object is created and returned.
  - String objects are "*immutable*" -> not directly changeable.

```
String finalText = "Java";
finalText.concat("and RPG");
System.out.println(finalText);
```

Answer: "Java"

- What is the result of following?
- Watch testing for equality of strings.
  - Do you think following code is valid?

```
if (text1 == text2)
```

Answer: No

- You have to use the equals method.

```
if (text1.equals(text2))
```

Strings

RPG o/c	RPG built-in	Description	Java Methods
CAT (or '+')		Concatenate two strings	concat(string) or '+' operator
SUBST	%SUBST	Extract a substring from a string.	substring(int start, int end) substring(int start)
SCAN	%SCAN	Scan for a substring	indexOf()
	%TRIM	Trim begin, end blanks	trim()
	%LEN	Return length of string	length()
XLATE		Translate a string	*Not Available*
CHECK		Check for characters	*Not Available*
CHECKR		Check in reverse	*Not Available*
	%TRIML	Trim leading blanks	*Not Available*
	%TRIMR	Trim trailing blanks	*Not Available*
	%CHAR	V4R2. Converts to string.	valueOf(datatype value) in String class
	%REPLACE	V4R2. Substring replacement	*Not Available*

Concatenation (1 of 2)

- RPG has traditional CAT op-code for appending one string to another.
- Java has formal concat method.

```
D  first                S                  10A     INZ('Mike')
D  last                 S                  10A     INZ('Smith')
D  name                 S                  20A     INZ(' ')
C        first       CAT          last:1        name
C        name        DSPLY
C                    MOVE         *ON           *INLR
```

RPG

```
String first, last, name;
first = "Mike ";
last  = "Smith";
name = first.concat(last);
System.out.println("The name is: " + name);
```

Java

Concatenation (2 of 2)

- Both RPG and Java also support the '+' operator for concatenation in expressions.
  - More intuitive than CAT op-code or concat method

```
C     EVAL      name = first  +  ' ' + last
```

RPG

```
name = first + " " + last;
```

Java

Substring

```
D*                                                  123456789012345678901234567890
DWhyJava          S              30A    INZ('Because Java is for RPG pgmrs')
D  first          S              10A    VARYING
D  second         S              10A    VARYING
D  third          S              10A    VARYING
D  sayWhat        S              30A
C     4             SUBST     WhyJava:9      first
C     6             SUBST     WhyJava:14     second
C     3             SUBST     whyJava:21     third
C                   EVAL      sayWhat = first + second + ' ' + third
C
C
```

RPG

```
public class Substring
{
    public static void main(String args[])
    {
        String whyJava, first, second, third, sayWhat;
        //          0123456789012345678901234567890123456789012345678901234
        whyJava = "Because Java is for RPG Programmers";
        first   = whyJava.substring(8,12);
        second  = whyJava.substring(13,19);
        third   = whyJava.substring(20,23);
        sayWhat = first + " " + second + " " + third;
        System.out.println(sayWhat);
    }
} // end class Substring
```

Java

First param = beginning index
Second param = ending index

Java is for RPG

Substring gotchas

- Important "gotchas" to note:
  - The parameters are zero-based, and not one-based as in RPG.
  - The second parameter is one past the actual ending position you want.
- In RPG IV you can use the %SUBST built-in function in expressions as follows:

```
%SUBST('RPG USERS':5:5)
%SUBST('RPG USERS':5)
```

USERS

USERS

Scanning for substrings

- RPG SCAN op-code scans for substring
  - Or you can use %SCAN built-in
- In Java, you use indexOf method
  - Or use lastIndexOf to search backwards!

```
D*                                                  12345678901234567890
Dstr                S              40A    INZ('Java is for RPG users')
Didx                S               3P 0
C      'RPG'          SCAN      str            idx
C      idx            DSPLY
```

13

RPG

Java

```
//                                  01234567890123456789 0
String str = new String("Java is for RPG users");
int idx = str.indexOf("RPG");
System.out.println("RPG occurs at: " + idx);
```

12

Trimming blanks

- RPG %TRIM built-in removes blanks from beginning and end
  - Or you can use %TRIML for beginning only, %TRIMR for end only
- In Java, you use trim method to remove blanks from both ends
  - No methods for removing from beginning or end only
  - You have to write your own (after you learn about StringBuffer)

RPG

```
D leftright        S              40A   INZ('      Java is for -
D                                       RPG users         ')
D temp             S              45A
C                    EVAL      leftright = %TRIM(leftright)
C     leftright      DSPLY
```

Java

```
String str = "    Java is for RPG users     ";
str = str.trim();
System.out.println("Trimmed: '" + str + "'");
```

String length

- To determine a character field's length in RPG, you can use the %LEN built-in function.
  - Returns the declared length of fixed size fields, current length of variable length fields (VARYING keyword)
- In Java, you can use the length() method.
  - Returns the length of the current contents of the String object

RPG

```
D inputString      S              30A   INZ('Java for RPG Programmers')
D varString        S              30A   VARYING
D                                       INZ('Java for RPG Programmers')
D len1             S               9  0
C                    EVAL      len1 = %LEN(inputString)
C      len1          DSPLY
C                    EVAL      len1 = %LEN(varString)
C      len1          DSPLY
```

30

24

Java

```
String inputString = "Java For RPG Programmers";
System.out.println(inputString.length());
```

24

XLATE operation

- RPG XLATE op-code translates characters.
  - Translates factor two source string
  - Based on from:to strings in factor one
  - All chars in from found in source are changed to chars in same position in to string.
  - Can optionally specify a starting position in source
- In Java there is no equivalent method.
  - You will write your own using the replace() method.

```
D  from                    C                    'GPR4'
D  to                      C                    'VAJA'
D  string                  S        4A          INZ('RPG4')
D  target                  S        4A
D*-------------------------------------------------------------
C        from:to      XLATE        string        target
C        target       DSPLY
```

RPG

JAVA

XLATE in Java

```
public static String xlate(String source,  String fromChars,
                           String toChars, int    start)
{
    String resultString;
    // minimal input error checking
    if (fromChars.length() != toChars.length())
      return new String("BAD INPUT!");
    if (start > source.length() || start < 0)
      return new String("BAD INPUT!");
    // first off, get the substring to be xlated...
    resultString = source.substring(start);
    // now, xlate each char in fromChars to same pos in toChars
    for (int i = 0; i < fromChars.length(); i++)
       resultString = resultString.replace(fromChars.charAt(i),
                                           toChars.charAt(i));
    // now append xlated part to non-xlated part
    resultString = source.substring(0,start) + resultString;
    return resultString;
} // end xlate method

public static String xlate(String source,  String fromChars,
                           String toChars)
{
    return xlate(source, fromChars, toChars, 0);
} // end xlate method two
```

Java

replace() replaces individual characters only

Testing XLATE in Java

- Test out new xlate method as follows:

```
public static void main(String args[])
{
    //              "01234567890123456789 0";
    String src  = "RPGP is for you Juys!";
    String from = "RPG";
    String to   = "JAV";

    System.out.println("Input string  : '" + src + "'");
    src = RPGString.xlate(src, from, to);
    System.out.println("Output string1: '" + src + "'");

    from = "J";
    to   = "G";
    src = RPGString.xlate(src, from, to, 16);
    System.out.println("Output string2: '" + src + "'");
}
```


```
Input string  : 'RPGP is for you Juys!'
Output string1: 'JAVA is for you Juys!'
Output string2: 'JAVA is for you Guys!'
```

Translating case

- In RPG, you have to use XLATE op-code.
- In Java, you use methods toXxxxCase()
  - Handles international characters too!

```
D Lower          C                    'abcdefghijklmnopqrstuvwxyz'
D Upper          C                    'ABCDEFGHIJKLMNOPQRSTUVWXYZ'
D WHAT           S              30A   INZ('Java is for rpg users')
C     WHAT         DSPLY
C     Lower:Upper  XLATE     WHAT          WHAT
C     WHAT         DSPLY
C                  MOVE      *ON           *INLR
```

RPG

```
String str = new String("Java is for RPG users");

str = str.toUpperCase();
System.out.println("String in Uppercase: " + str);
str = str.toLowerCase();
System.out.println("String in lowercase: " + str);
```

Java

CHECK in RPG

- RPG has support to check for existence of characters:
  - CHECK op-code to check from left
  - CHECKR op-code to check from right
- It verifies that all characters in search string (factor one) are in base string (factor two)

```
D  numbers                C                                  '0123456789'
D  base                            7A
D*-------------------------------------------------------------------
C                     MOVE          '*22300*'      base
C        numbers      CHECK         base:2         pos      3 0
C        pos          DSPLY
                                  7
C        numbers      CHECKR        base:6         pos
C        pos          DSPLY
                                  1
C                     MOVE          *ON            *INLR
```

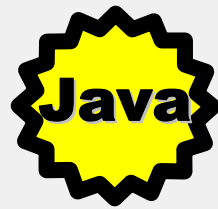
CHECK in Java

- Java has no equivalent to RPG's CHECK or CHECKR op-codes.
- You write your own static methods.
  - check() that takes starting position
  - check() that does not take starting position (default is start)
  - checkr() that takes starting position
  - checkr() that does not take starting position (default is end)
- You will use:
  - charAt(int index) method to extract each char of "base" string
  - indexOf(int index) method to search "search" string for extracted character of base (returns -1 if not found)
- Remember, indexing starts at zero in Java

CHECK method in Java

```
public class RPGString
{
    public static int check(String search, String base,
                            int start)
    {
        // minimal error checking
        if (start >= base.length() || start < 0)
          return -2;
        // scan each char of base for match in search...
        for (int idx = start; idx < base.length(); idx++)
           if (search.indexOf(base.charAt(idx)) == -1)
             return idx;
        // return constant indicating match found for all
        return -1;
    } // end check method

    public static int check(String search, String base)
    {
        return check(search, base, 0);
    } // end check method two
```

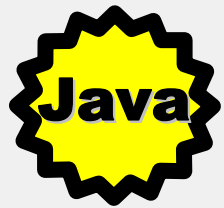


CHECKR method in Java

```
public static int checkR(String search, String base,
                                 int start)
   {
       // minimal error checking
       if (start >= base.length() || start < 0)
         return -2;

       // scan each char of base for match in search...
       for (int idx = start; idx >= 0; idx--)
          if (search.indexOf(base.charAt(idx)) == -1)
            return idx;
        // return constant indicating match found for all
       return -1;
   } // end checkR method

   public static int checkR(String search, String base)
   {
       return checkR(search, base, base.length()-1);
   } // end checkR method two
 } // end RPGString class
```



Testing CHECK in Java

```
public class TestCheck
{
    public static void main(String args[])
    {
        String digits = "0123456789";
        String test   = "*22300*";
        int    result;
        // check if test is has only digits
        result = RPGString.check(digits, test);
        System.out.println("result is: " + result);
        result = RPGString.check(digits, test, 1);
        System.out.println("result is: " + result);

        result = RPGString.checkR(digits, test);
        System.out.println("result is: " + result);
        result = RPGString.checkR(digits, test, 5);
        System.out.println("result is: " + result);
    } // end main method
} // end TestCheck class
```


```
result is: 0
result is: 6

result is: 6
result is: 0
```

More string methods

Method	Description
`compareTo(String)`	Compares two strings lexicographically
`copyValueOf(char[],int,int)`	Returns a String equivalent to specified character array
`endsWith(String)`	Tests if this string ends with the specified suffix
`equals(Object)`	**Compares this string to the specified object**
`equalsIgnoreCase(String)`	**Compares this String to another, ignoring case**
`getBytes()`	Convert this String into a byte array
`getChars(int,int,char[],int)`	Copies characters into character array, starting at offset
`hashCode()`	Returns a hashcode for this string
`intern()`	Returns canonical representation for efficient comparisons
`regionMatches(boolean, int, String, int, int) ...`	Tests if two string regions are equal
`charAt(int)`	**Returns character at given index**
`startsWith(String)`	Tests if this string starts with the specified prefix
`toCharArray()`	Converts this string to a new character array
`toLowerCase(Locale)`	**Converts this String to lower case**
`toUpperCase(Locale)`	**Converts this String to upper case**
`valueOf(xxx)`	**Converts primitive data type value to a string**

StringBuffer

- All String methods, such as concat, toUpperCase, and replace, do not affect the string object.
  - They simply return a new string object.
- String objects are immutable.
  - No way to change original object, only to get a new one
  - Original string is swept away by the garbage collector.
  - They can have performance implications.
- Java supplies a second string class called StringBuffer, which is mutable!
  - String and StringBuffer are completely independent.
  - If you need to dynamically change the strings in your methods, use StringBuffer class for better performance.

Using StringBuffer

- StringBuffer objects have no built-in language support.
  - Must use **new** operator
  - Cannot use "+" operator; must use append(...) method
- Need to do a lot of manipulation on a String?
  - Convert to StringBuffer using constructor
  - After manipulations convert back to String using toString()
  - Much better performance for heavy manipulation

```
public String workOnString(String input)
{
    StringBuffer workString = new StringBuffer(input);
    // do manipulation work on the workString variable
    return workString.toString();
}
```

StringBuffer appends

- To concatenate with StringBuffer, use the append method.

```
StringBuffer quotedName = new StringBuffer("George");
quotedName.append(" and ").append("Phil");
```

- You can append any primitive value.

```
boolean flag = true;
StringBuffer output = new StringBuffer("flag value = ");
output.append(flag);
System.out.println(output);
```

flag value = true

- You can also append objects!
  - Their toString() method is called to convert to a string.

StringBuffer inserts

- You can also insert strings in middle of StringBuffer objects, using insert method:

```
StringBuffer quotedName = new StringBuffer("Grge");
quotedName.insert(1,"eo");
```

"George"

- You can insert any primitive value:

```
boolean flag = true;
StringBuffer output = new StringBuffer("its , no?");
output.insert(4,flag);
System.out.println(output);
```

"its true, no?"

- You can also insert objects.
  - Their toString() method is called to convert to a string.

StringBuffer characters

- You can change characters in the middle of a StringBuffer object using setCharAt method:

```
StringBuffer quotedName = new StringBuffer("Goorge");
quotedName.setCharAt(1,'e');
```

"George"

- You can also extract any character using charAt method:

```
StringBuffer quotedName = new StringBuffer("George");
char firstChar = quotedName.charAt(0);
System.out.println(firstChar);
```

'G'

- You can also extract a substring.
  - Use getChars() method, but it extracts into a char array

StringBuffer capacity

- Internally, StringBuffer objects have a buffer.
- To get best performance, you can set the buffer's default and current *capacity* (size).
  - The default is 16 characters.
- You can set it when instantiating:

```
StringBuffer largeString = new StringBuffer(255);
```

- You can also set it with ensureCapacity method.
  - Given a number, will increase buffer size by 2*current-capacity+2 if current-capacity is less than that number
  - And query it with capacity() method

StringBuffer length

- While capacity is the size of the buffer, length() returns current number of characters in buffer:
  - Length <= capacity
- Length is usually implicitly set by the contents of the StringBuffer object.
  - However, it can be explicitly set with setLength()
  - It will pad with hex zeros if greater than current length (and grow capacity if it has to).
  - It will truncate contents if less than current length.

StringBuffer example

- Recall that RPG can trim leading or trailing blanks by using %TRIML or %TRIMR built-in functions

```
D input          S          16A   INZ('   Java for U   ')
D result         S          16A
C                  EVAL     result = %TRIML(input) + '.'
C     result       DSPLY    'Java for U   .'

C                  EVAL     result = %TRIMR(input) + '.'
C     result       DSPLY    '   Java for U.'
```

- Java strings have only <u>trim</u> method.
  - No way to trim leading-only or trailing-only blanks
  - You will write your own methods using **StringBuffer** class.

Example: trimr

- Our own "trim right" method:


```
public class RPGString
{
   public static String trimr(String input)
   {
       if (input.length() == 0) // error checking
         return input;

       StringBuffer temp = new StringBuffer(input);
       int idx;
       // find last non-blank character
       for (idx = temp.length()-1;
            (idx >= 0) && (temp.charAt(idx) == ' ');
            idx--);
       // truncate string
       if (idx <= 0)
         idx = 0;
       temp.setLength(idx+1);
       return temp.toString();
   } // end trimr method
} // end RPGString class
```

Example: triml

- Our "trim left" method:

```
public static String triml(String input)
{
    if (input.length() == 0) // error checking
      return input;

    StringBuffer temp1 = new StringBuffer(input);
    int idx, idx2;

    // find first non-blank character
    for (idx = 0;
          (idx<temp1.length()) &&
          (temp1.charAt(idx)==' ');
         idx++);
    // copy characters to new object
    int newSize = temp1.length() - idx;
    StringBuffer temp2 = new StringBuffer(newSize);
    for (idx2 = 0; idx2 < newSize; idx2++, idx++)
       temp2.append(temp1.charAt(idx));

    return temp2.toString();
} // end triml method
```

Example: Testing

- A little program to test your trimr and triml new methods:

```
public class TestTrim
{
   public static void main(String args[])
   {
       String test = "  Java is for RPG Programmers  ";
       String result;

       System.out.println("initially: '" + test + "'");
       result = RPGString.trimr(test);
       System.out.println("result is: '" + result + "'");
       result = RPGString.triml(result);
       System.out.println("result is: '" + result + "'");
   } // end main method
} // end TestTrim class
```

```
initially: '  Java is for RPG Programmers  '
result is: '  Java is for RPG Programmers'
result is: 'Java is for RPG Programmers'
```

Tokenizing

- Often you need to parse a string into individual words
  - How could you do this in RPG??


```
D formula          C                    'A * 2 / 3 - Nu
D tempstr          S              10A
D start            S               2P 0 INZ(1)
D end              S               2P 0 INZ(0)
C                    DOW        (start <= %LEN(formula))
C                    EVAL       end = %SCAN(' ':formula:start)
C                    IF         end = 0
C                    EVAL       end = %LEN(formula)+1
C                    ENDIF
C                    EVAL       tempstr=
C                               %SUBST(formula:start:end-
start)
C      tempstr       DSPLY
C                    EVAL       start=end+1
C                    ENDDO
```

StringTokenizer class

- Java supplies a class StringTokenizer in the java.util package.
- Given a string, it has methods to iterate through it one word at a time:
  - hasMoreTokens returns true if there are more words
  - nextToken returns the next word in the string

```
String sample = "Java for U";
StringTokenizer words = new StringTokenizer(sample);
while (words.hasMoreTokens())
  System.out.println("next word = " + words.nextToken());
```

easy!

Topics covered

- Introduction
- Strings
- Concatenating strings
- Substrings
- Scanning strings
- Trimming strings
- Translating strings
- Checking strings
- StringBuffer and StringTokenizer

Unit summary

Having completed this unit, you should be able to:

- Represent a character string in Java
- Perform operations on the string, such as concatenation, substring extraction and location, and space trimming
- Create comparable Java capability from common RPG built-in functions and operations that manipulate strings