CS 1301 – Ch 8, Handout 2

This section discusses the *split* and *match* methods of the String class, regular expressions, and creating objects from strings, and command line arguments.

The split Method

1. The `split( delimiter : string ) : string[]` method of the *String* class is useful for *parsing* a string, for breaking it into an array of substrings based on the delimiter. Sometimes these array items are called *tokens*/

```
values = "24.33,33.44,66.7778,99.30,77.213";
delimiter = ",";
nums = values.split( delimiter );
System.out.println( Arrays.toString( nums ) );
```

which produces:

```
[24.33, 33.44, 66.7778, 99.30, 77.213]
```

an array of Strings.

However, what if there happened to be a space in between a comma and a number:

```
values = "24.33, 33.44,66.7778,99.30,77.213";
delimiter = ",";
nums = values.split( delimiter );
System.out.println( Arrays.toString( nums ) );
```

which produces:

```
[24.33,  33.44, 66.7778, 99.30, 77.213]
```

Notice that the second element is: `' 33.44'.` This may, or may not be a problem.

2. We can use anything as a string delimiter:

   a. Example 1

```
values = "Alton#Quincy#Van Horton";
delimiter = "#";
names = values.split( delimiter );
System.out.println( Arrays.toString( names ) );
```

which produces:

```
[Alton, Quincy, Van Horton]
```

b. Example 2

```
values = "AltonRedQuincyRedVan Horton";
delimiter = "Red";
names = values.split( delimiter );
System.out.println( Arrays.toString( names ) );
```

which produces:

```
[Alton, Quincy, Van Horton]
```

The split Method and Regular Expressions

1. Using Strings as delimiters is very useful and powerful, although they enforce rigid conformity. We can introduce more flexibility by using the `split( regularExpression : string ) : string[]`. A regular expression can look for a pattern of strings. For instance, suppose we wanted to parse a set of numbers where numbers are separated by a comma and 1 or more spaces.

```
values = "24.33,33.44,    66.7778, 99.30, 77.213";
regExp = ",\\s*";
nums = values.split( regExp );
```

which produces:

```
num='24.33','33.44','66.7778','99.30','77.213'
```

The regular expression: ,\\s* says to use a comma followed by zero (*) or more spaces (\\s) as the delimiter.

2. Example 2

```
values = "24.33   33.44 66.7778 99.30  77.213";
regExp = "\\s+";
nums = values.split( regExp );
```

which produces:

```
num='24.33','33.44','66.7778','99.30','77.213'
```

The regular expression: `\\s+` says to use one or more (+) spaces as a delimiter.

3. Example 3

```
values = "24.33,33.44:66.7778?99.30;77.213";
regExp = "[,:;?]";
nums = values.split( regExp );
```

which produces:

```
num='24.33','33.44','66.7778','99.30','77.213'
```

The regular expression: `[,:;?]` says to use a comma, colon, semicolon, or question mark as a delimiter.

4. Example 4 – Read the words in a paragraph:

```
values = "The time, now, has come. Thus: be ready! OK?";
regExp = "[\\s.,:!?]\\s*";
words = values.split( regExp );
```

which produces:

```
words='The','time','now','has','come','Thus','be','ready','OK',
```

The regular expression: `[\\s.,:!?]\\s*` says to use a space, period, comma, colon, semicolon, or question, followed by any number of spaces as a delimiter.

The match Method and Regular Expressions

5. The *match* and *replaceAll* method can be used with regular expressions:

```
"The house is red".matches( "The.*" ) );
"The house is blue".matches( "The.*" ) );
"The house is red".matches( "The.*red" ) );
"The red garden".matches( "The.*red.*" ) );
"The house is here".matches( ".*house.*" ) );
"The house is here".matches( "(.*house.*)|(.*here)" ) );
"Dad is here".matches( "(.*house.*)|(.*here)" ) );
```

All these produce the value: *true*.

```
"The house is here".matches( ".*[h,i].*" ) ); // true

String s1 = "The house is here.";
s1 = s1.replaceAll("[a,e,i,o,u]", "v" );
System.out.println( s1 );

s1 = "cc3asd45asdf";
s1 = s1.replaceAll("[0-9]", "ZZ" );
System.out.println( s1 );
```

Examples

1. Suppose that we want to read a string from the user and build a Student object based on the contents of the string. For instance, suppose the user enters: dave 22 89 99 87. The first value is the name, the second is the age, and the remaining values are test scores. First, consider this Student class which takes such a string a parses it to create the object.

```
public class Student
{
    private String name;
    private int age;
    private int[] scores;
    private int numScores;

    public Student( String strStudent )
    {
       String[] strStudentArray = strStudent.split( "\\s+" );

       this.name = strStudentArray[0];
       this.age = Integer.parseInt( strStudentArray[1] );

       numScores = strStudentArray.length - 2;

       scores = new int[ numScores ];

       for( int i=0; i<numScores; i++ )
            scores[i] = Integer.parseInt( strStudentArray[i+2] );

    }

    public String getName() { return name; }
    public int getAge() { return age; }
    public int getScore( int i ) { return scores[i]; }
    public int getNumScores() { return numScores; }
}
```

2. Now, consider this driver for the *Student* class.

```
public static void main(String[] args)
{
      Scanner s = new Scanner( System.in );
      String strInput;

      System.out.println( "Enter the data for a Student: " +
                          "name age score1 s2 ...");
      strInput = s.nextLine();

      Student st = new Student( strInput );

      printStudent( st );


}

public static void printStudent( Student s )
{
      System.out.printf( "name=%s, age=%d, scores=",
                  s.getName(), s.getAge() );

      for( int i=0; i<s.getNumScores(); i++ )

            System.out.printf( "%d, ", s.getScore(i) );

      System.out.println();
}
```

3. Now, suppose that we want to enter a bunch of students this way.:

```
Student[] students = new Student[100];
int numStudents = 0;

Scanner s = new Scanner( System.in );
String strInput;

while( numStudents <= 100 )
{
     System.out.println( "Enter the data for a Student: " +
                         "name age score1 s2 ...");

     strInput = s.nextLine();

     if ( strInput.equals("") )
        break;
     else
     {
        Student st = new Student( strInput );
        students[numStudents++] = st;
     }
}

for( int i=0; i<numStudents; i++ )
     printStudent( students[i] );
```

4. Suppose we want the user to enter data to create a circle or a rectangle. To specify a circle with a radius of 5, the user will enter: *circle 5.0*. For a rectangle with length 3 and width 4.5: *rectangle 3 4.5*. Thus, tokenize the string and then check to see if the first token is "circle" or "rectangle". If it is "circle", then you know there is only one more token which contains the radius.

```
int numStudents = 0;
String regExp;
String[] strShapeArray;
String shapeType;
Circle c;
Rectangle r;

Scanner s = new Scanner( System.in );
String strInput = "y";

while( strInput.equals("y") )
{
   System.out.println( "Enter the data for a shape");
   System.out.println( "e.g.:circle 12.5" );
   System.out.println( "e.g.:rectangle 6.0 12.0" );

   strInput = s.nextLine();
   regExp = "\\s+";
   strShapeArray = strInput.split( regExp );
   shapeType = strShapeArray[0];

   if( shapeType.equals("circle") )
   {
        c = new Circle( Double.parseDouble( strShapeArray[1] ) );
        System.out.println( c.toString() );
   }
   else if ( shapeType.equals("rectangle") )
   {
        r = new Rectangle( Double.parseDouble( strShapeArray[1] ),
                           Double.parseDouble( strShapeArray[2] ) );
        System.out.println( r.toString() );

   }

   System.out.println( "Want to enter another shape (y or n)?");
   strInput = s.nextLine();
  }
 }
```

.

The corresponding *Circle* and *Rectangle* classes are:

```
public class Circle
{
     private double radius;

     public Circle( double radius )
     {
          this.radius = radius;
     }

     public double getRadius() { return radius; }
     public double getArea() { return Math.PI * radius * radius; }

     public String toString()
     {
          String msg = String.format( "Circle: radius=%.2f, " +
                                      "area=%.2f",
                                      getRadius(), getArea() );

          return msg;
     }
}

public class Rectangle
{
     private double length;
     private double width;

     public Rectangle( double lenght, double width )
     {
          this.length = length;
          this.width = width;
     }

     public double getLength() { return length; }
     public double getWidth() { return width; }
     public double getArea() { return length * width; }
     public String toString()
     {
          String msg = String.format( "Rectangle: length=%.2f, " +
                                      "width=%.2f, " +
                                      "area=%.2f",
                                      getLength(),
                                      getWidth(), getArea() );
          return msg;

     }
}
```

Reading from the Command Line

1. Java programs can take *command line arguments.* For instance, when we run a Java program we do this:

```
java Circle
```

   We could pass information into the program like this:

```
java Circle 3
```

   Values that are passed into a program are caught in the String[] args array (or whatever name you give it). For instance, in the example above, we could write:

```
int radius = Double.parseDouble( args[0] )
```

2. Java parses the command line with one or more spaces as the delimiter.

3. In JGrasp, choose *Build, Run Arguments* and a text box will appear at the top of the window your class is in.

4. Example: Now, suppose that you want to read the data for the previous example from the commandline. For instance: java ShapeCalculator circle 12.0. In this case, we use the args[] array that is passed to main.

```
int numStudents = 0;
String regExp;
String[] strShapeArray;
char shapeType;
Circle c;
Rectangle r;

shapeType = args[0].charAt(0);

switch( shapeType )
{
     case 'c' :
          c = new Circle( Double.parseDouble( args[1] ) );
          System.out.println( c.toString() );
          break;

     case 'r' :
          r = new Rectangle( Double.parseDouble( args[1] ),
                             Double.parseDouble( args[2] ) );
          System.out.println( r.toString() );
          break;

     default:
          System.out.println( "unrecognized shape" );


}
```