Text processing
**Text processing**

- **text processing**: Examining, editing, formatting text.
  - Text processing often involves *for* loops that examine the characters of a string one by one.

- Two data types involved

<table>
<thead>
<tr>
<th>char</th>
<th>String</th>
</tr>
</thead>
<tbody>
<tr>
<td>Represents <strong>individual characters</strong></td>
<td>Represents <strong>sequences of characters</strong></td>
</tr>
<tr>
<td>Primitive type</td>
<td>Object type (i.e., not primitive)</td>
</tr>
<tr>
<td>Written with single quotes</td>
<td>Written with double quotes</td>
</tr>
</tbody>
</table>

- e.g.:
  - ‘T’
  - ‘t’
  - ‘3’
  - ‘%’
  - ‘\n’

- e.g.:
  - “We the people”
  - “1. Twas brillig, and the slithy toves\n”
  - “”
  - “T”
Characters

- **char**: A *primitive* type representing single characters.

- Individual characters inside a *String* are stored as *char* values.

- **Literal char values** are surrounded with apostrophe (single-quote) marks, such as 'a' or '4' or '
' or '\'

- Like any other type, you can create variables, parameters, and returns of type *char*.

  ```java
  char letter = 'S';
  System.out.println(letter);    // prints S
  ```
Strings

- String: an object type for representing **sequences** of characters
  - Sequence can be of length 0, 1, or longer
  - Each element of the sequence is a **char**
  - We write strings surrounded in double-quotes
  - We can declare, initialize, assign, and use **String** variables in expressions just like other data types

```java
String s = "Hello, world\n";       // declare, init
System.out.println(s);            // use value
s = s + "I am your master\n";      // concatenate
                                 // and assign
```
String Methods

Unlike primitive types, **object types can have methods**.

Here is a list of methods for strings:

<table>
<thead>
<tr>
<th>Method name</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>charAt(&lt;i&gt;index&lt;/i&gt;)</td>
<td>returns the character at the given index</td>
</tr>
<tr>
<td>indexOf(&lt;i&gt;str&lt;/i&gt;)</td>
<td>returns the index where the start of the given string appears in this string (-1 if not found)</td>
</tr>
<tr>
<td>length()</td>
<td>returns the number of characters in this string</td>
</tr>
<tr>
<td>substring(&lt;i&gt;index1, index2&lt;/i&gt;)</td>
<td>returns the characters in this string from &lt;i&gt;index1&lt;/i&gt; up to, but not including, &lt;i&gt;index2&lt;/i&gt;</td>
</tr>
<tr>
<td>toLowerCase()</td>
<td>returns a new string with all lowercase letters</td>
</tr>
<tr>
<td>toUpperCase()</td>
<td>returns a new string with all uppercase letters</td>
</tr>
</tbody>
</table>
The `charAt` method

- The characters of a string can be accessed using the `String` object's `charAt` method.
- String indices start at 0.

```java
String word = "cola";
char firstLetter = word.charAt(0);
if (firstLetter == 'c') {
    System.out.println("C is for cookie!");
}
```

<table>
<thead>
<tr>
<th><code>charAt(i)</code>:</th>
<th>‘c’</th>
<th>‘o’</th>
<th>‘l’</th>
<th>‘a’</th>
</tr>
</thead>
<tbody>
<tr>
<td>Index i:</td>
<td>0</td>
<td>1</td>
<td>2</td>
<td>3</td>
</tr>
</tbody>
</table>

Starts at 0!
Calling string methods

- Let \( s \) be a variable of type \texttt{String}
- General syntax for calling a \texttt{String} method:

  \[
  s.<\text{method}>(\text{<args>})
  \]

- Some examples:

```java
String s = "Cola";
int len = s.length(); // len == 4
char firstLetter = s.charAt(0); // 'C'
int index = s.indexOf("ol"); // index == 1
String sub = s.substring(1, 3); // "ol"
String up = s.toUpperCase(); // "COLA"
String down = s.toLowerCase(); // "cola"
```
Fun with char!

- **char values can be concatenated with strings.**
  ```java
  char initial = 'P';
  System.out.println(initial + " . Diddy");
  ```

- **You can compare char values with relational operators.**
  - 'a' < 'b' and 'Q' != 'q'
  - **Caution:** You should **NOT** use these operators on a String!

- **Example:**
  ```java
  // print the alphabet
  for (char c = 'a'; c <= 'z'; c++) {
      System.out.print(c);
  }
  ```
char vs. String

- 'h' is a char
  ```java
  char c = 'h';
  ```
  - char values are **primitive**; you cannot call methods on them
  - can't say `c.length()` or `c.toUpperCase()`

- "h" is a String
  ```java
  String s = "h";
  ```
  - Strings are objects; they contain methods that can be called
  - can say `s.length()` 1
  - can say `s.toUpperCase()` "H"
  - can say `s.charAt(0)` 'h'
Numbers vs Strings

- **345 is an int**
  ```java
  int i = 345;
  ```
- **int values are primitive; you cannot call methods on them**
- **CAN perform arithmetic:**
  ```java
  i = i * 2; // i==690
  ```
- **CANNOT say** `i.length()` **or** `i.charAt(0)`

- **“345” is a String**
  ```java
  String s = "345";
  ```
- **Strings are objects; they contain methods that can be called**
  - **can say** `s.length()` **// returns 3**
  - **can say** `s.charAt(1)` **// returns ‘4’**
  - **CANNOT perform arithmetic:**
    ```java
    s = s * 2; // ERROR!
    ```
More text processing: Comparing strings

- Objects (such as `String`) should be compared for equality by calling a method named `equals`.

- Example:
  ```java
  Scanner console = new Scanner(System.in);
  System.out.print("What is your name? ");
  String name = console.next();
  if (name.equals("Barney")) {
      System.out.println("I love you, you love me,");
      System.out.println("We're a happy family!");
  }
  ```
More text processing: Comparing strings

- There are more methods of a `String` object that can be used in `<test>` conditions.

<table>
<thead>
<tr>
<th>Method</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td><code>equals(str)</code></td>
<td>whether this string contains exactly the same characters as the other string</td>
</tr>
<tr>
<td><code>equalsIgnoreCase(str)</code></td>
<td>whether this string contains the same characters as the other, ignoring upper- vs. lowercase differences</td>
</tr>
<tr>
<td><code>startsWith(str)</code></td>
<td>whether this string’s beginning matches the argument</td>
</tr>
<tr>
<td><code>endsWith(str)</code></td>
<td>whether this string’s end matches the argument</td>
</tr>
</tbody>
</table>
Comparing strings: Examples

Hypothetical examples, assuming the existence of various String variables:

- if (title.endsWith("M.D.")) {
      System.out.println("What's your number?");
  }

- if (fullName.startsWith("Giorgio")) {
      System.out.println("When are you retiring?");
  }

- if (lastName.equalsIgnoreCase("lumBerg")) {
      System.out.println("I need your TPS reports!");
  }

- if (name.toLowerCase().indexOf("sr.") >= 0) {
      System.out.println("You must be old!");
  }
Data type conversion

Sometimes, we want to change between types.

For example, 2640 feet = x in miles?

```java
int x = 2640;
int y = 5280;

return x / y;  // integer arithmetic
```

This returns 0 if 3000 and 5280 are both ints.
Sometimes, we want to change between types.

For example, 3000 feet = x in miles?

```java
int x = 2640;
int y = 5280;
double z = (double) x;

return z / y;  // double arithmetic
```
An expression like this:

\[(\text{<type>}) \text{ <expression>}\]

is called a *typecast*. It converts the expression into an equivalent expression of type \text{<type>}.

**Examples:**

\begin{align*}
\text{(double)} \ 2640 & \ \text{==} \ 2640.0 \\
\text{(int)} \ 32.7 & \ \text{==} \ 32 \\
\text{(int)} \ \text{‘A’} & \ \text{==} \ ?? \\
\text{(char)} \ 97 & \ \text{==} \ ?? \\
\text{(String)} \ \text{‘A’} & \ \text{==} \ ?? \\
\text{(String)} \ 27 & \ \text{==} \ ??
\end{align*}
Typecasting

An expression like this:

\[(\text{<type>}) \text{ <expression>}\]

is called a *typecast*. It converts the expression into an equivalent expression of type \(\text{<type>}\).

**Examples:**

- \(\text{(double) 2640} \quad \text{==} \quad 2640.0\)
- \(\text{(int) 32.7} \quad \text{==} \quad 32\)
- \(\text{(int) ‘A’} \quad \text{==} \quad 65\)
- \(\text{(char) 97} \quad \text{==} \quad ‘a’\)
- \(\text{(String) ‘A’} \quad \text{==} \quad \text{TypeCastException}\)
- \(\text{(String) 27} \quad \text{==} \quad \text{TypeCastException}\)
What about String typecasts?

The (String) typecasts don’t work with primitive types, like char, int, and double.

However, there are easy ways to convert to and from strings:

**Converting to strings:**
- `int x` to `String`: `x + ""` or `"" + x`
- `double x` to `String`: `x + ""` or `"" + x`
- `char, boolean, or float x` to `String`: `same thing`
What about String typecasts?

The (String) typecasts don’t work with primitive types, like char, int, and double.

However, there are easy ways to convert to and from strings:

Converting from strings:
String s to int: Integer.parseInt(x)
String s to double:
  Double.parseDouble(x) ...
String s to char: Character.parseChar(x)
Reading in Strings

Remember the `Scanner` method called `nextInt()`?

You can also use Scanners to read in Strings with the `next()` and `hasNext()` methods.

Example:

Please enter your name: **Alexander Pieter Yates**

Name 1 has 9 letters
Name 2 has 6 letters
Name 3 has 5 letters
import java.util.Scanner;

public class NameLength {
    public static void main(String[] args) {
        Scanner scan = new Scanner(System.in);
        System.out.print("Please enter your name");
        int i = 1;
        while (scan.hasNext()) {
            String s = scan.next();
            int length = s.length();
            System.out.println("Name "+i+" has "+length+" letters");
            i++;
        }
    }
}
## Scanner class (so far)

<table>
<thead>
<tr>
<th>return</th>
<th>method</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>int</td>
<td>nextInt()</td>
<td>Reads the next token, converts it to an int (if possible), and returns the int value.</td>
</tr>
<tr>
<td>double</td>
<td>nextDouble()</td>
<td>Reads the next token, converts it to a double (if possible), and returns the double value.</td>
</tr>
<tr>
<td>String</td>
<td>next()</td>
<td>Reads the next token, and returns it.</td>
</tr>
<tr>
<td>boolean</td>
<td>hasNext()</td>
<td>Returns true if there are more tokens.</td>
</tr>
</tbody>
</table>
if/else Variation
if/else Variation

char grade;
....  // compute the grade
if(grade=="A" || grade=="a") {
    System.out.println("Great job!");
} else if(grade=="B" || grade=="b") {
    System.out.println("Very nice job.");
} else if(grade=="C" || grade=="c") {
    System.out.println("Good job.");
} else if(grade=="D" || grade=="d") {
    System.out.println("Keep at it.");
} else {
    System.out.println("Work harder!");
}
switch statement

cchar grade;
....  // compute the grade
switch(grade) {
    case 'A','a': System.out.println(“Great job”);
    break;
    case 'B','b': System.out.println(“Very nice job”);
    break;
    case 'C','c': System.out.println(“Good job”);
    break;
    case 'D','d': System.out.println(“Keep at it”);
    break;
    default: System.out.println(“Work harder!”);
}
switch statement

char grade;
.... // compute the grade
switch(grade) {
    case 'A','a': System.out.println("Great job"); break;
    case 'B','b': System.out.println("Very nice job"); break;
    case 'C','c': System.out.println("Good job"); break;
    case 'D','d': System.out.println("Keep at it"); break;
    default: System.out.println("Work harder!");
}
switch syntax

switch(<discrete variable or expression>) {  
case <value 1>:  <statements;>  
    break;  
case <value 2>:  <statements;>  
    break;  
...

default:  <statements;>
}  

Note: The discrete variable may be of type byte, short, int, long, or char (no doubles, floats, Strings, or other object types).
Common Programming Error: switch

// Don't forget the break statement!
char grade;
.... // compute the grade
switch(grade) {
    case 'A','a': System.out.println("Great job");
                  break;
    case 'B','b': System.out.println("Very nice job");
                  // Where's the break?
    case 'C','c': System.out.println("Good job");
                  break;
    case 'D','d': System.out.println("Keep at it");
                  break;
    default:      System.out.println("Work harder!");
}