Character Data

- Computers represent information using binary digits (bits)
- The entity a bit pattern represents depends on the data type, which tells the computer how to interpret the bits
- For example, the bit pattern 0000 0000 0100 0001 might represent the integer (**int**) 65 or the character (**char**) 'A'
- Java uses the Unicode standard for representing character data, which is based on ASCII
- One can convert between an **int** and a **char** as follows:

```
int n = 65;
char c = (char) n;
char d = 'A';
int m = (int) d;
```

- A **char** literal uses the single quote character ' as a delimiter

More on using variables of type **char**

- One can print a variable of type **char** using **print** or **println**

```
char c = 'A'; // the variable c contains
   the character A
println(c); // produces A
```

- One can print a variable of type **char** using the **%c** specifier in **printf**

```
char c = 'A'; // the variable c contains
   the character A
System.out.printf("%c", c); // produces A
```

- To print a character, it must have a glyph (pictorial representation) in the current font
- Not all characters have an associated glyph

Special **char** values

- The backslash character '\' is a meta character, imparting special meaning to the character that follows
- Here are some common usages
  - **\n** represents a newline character
  - **\r** represents a carriage return
  - **\t** represents a tab
  - **\b** represents a backspace
  - \\ represents a backslash
- The following are equivalent

```
println('A');

print('A'); print('\n');

System.out.printf("%c\n", 'A');
```

The **String** data type

- It is often useful to have program that process text
- Java has a specific data type for this called a **String**
- A **String** is a special data entity called an *Object*
- Objects generally encapsulate a set of data or attributes and a set of functions
- A **String** is a sequence of **char** values together with functions that can manipulate it
- The functionality of predefined objects, like **String**, is found in the Java API (application programming interface)
- See the **String** API
- Technically the definition of a **String** is called the class definition
- A particular **String** entity is called an Object
- A variable of type **String** is called a reference

Using the **String** data type

- A **String** literal uses the double quote character " as a delimiter

```
String s; // s refers to a String object
System.out.println(s); // null
s = "computer"; // s refers to the
                // object "computer"
System.out.println(s);
```

- The variable **s** is a reference variable, it refers to a location in memory storing a **String**
- A **String**, like any object, can have the special value **null**, which means the reference to which the variable is referring is undefined.
- The **new** keyword can be used to create a **String** object, but is optional

Using the **String** methods

- Java uses a dot to indicate attributes and methods associated with an object
- There are many methods defined for **String**s as seen in the API
- Here are two important ones
  - **length()** returns the length
  - **charAt(int i)** returns the character at index **i**
- Here is a way to output a **String** one character at a time

```
String s = "computer";
for (int i = 0;  i < s.length();  i++) {
   println(s.charAt(i));
   }
```

- We can think of the methods as a mechanism for querying the object with regard to its state.
- Object methods always use parentheses

Concatenating **Strings**

- The **concat()** method will create a new **String** object

```
String s1 = "cob";
String s2 = "web";
println(s1.concat(s2));
```

- Because this is so common, Java allows the equivalent using the + operator, which works left to right

```
String s1 = "cob";
String s2 = "web";
println(s1+s2);
```

- The + operator can be used to combine **Strings** with other data types

String equality

- When comparing **String** references == operator tests to see if the references refer to the same object
- When comparing **String** references, the **equals()** method tests to see if the object references refer to objects containing the same *value*

```
String s1 = "cob";
String s2 = "web";
String s3 = s1+s2;
println(s1==s2);  // false
println(s1.equals(s2));  // false
String s4 = s3;
println(s3==s4);  // true
String s5= "cobweb";
println(s3==s5);  // false
println(s3.equals(s5));  // true
```