Strings & StringBuffer

- String:
  - Java library provides String class
  - stores sequence of character data
  - Example:

```
public static void main(String[] args)
{         String s;
          s = “Comp182";
          ...
          s = “Comp282";
}
```

Strings & StringBuffer

- String initialization
  - Strings can be initialized when defined
  - Example:

```
public static void main(String[] args)
{
        String s = "hollywood";
        ...
}
```

Strings & StringBuffer

- String concatenation
  - Operator + used for concatenation
  - Example:

```
public static void main(String[] args)
{
        String s1 = “Comp182";
        String s2;
        s2 =s1 + “/L”;
        ...
}
```

Strings & StringBuffer

- String output
  - println method used to print strings
  - Example:

```
public static void main(String[] args)
  {
        String s = “Comp182L";
        System.out.println(s);
        ...
  }
```

Strings & StringBuffer

- String concatenation with a number
  - Primitive-type data can be concatenated to String
  - Primitive-type data are converted to String data
  - Convenient for printing
  - Example:

```
public static void main(String[] args)
  {
            boolean b = True;
            String s = “Are you single? "+”/n”+ b;
            System.out.println (s) ;
  }
```

Strings & StringBuffer

- String assignment
  - Strings are constant
  - Hence, never change after creation
  - Assignment creates new object
  - Example:

```
String s;        // new address is created.
s = “hi";        //create a new object
s = “bye";       //create a new object
```

  - Two objects are created. Only one is accessible by s.

Strings & StringBuffer

- String assignment and parameters
  - change inside method does not affect original string
  - Example:

```
class Semantic
  {
  void add(String t)
     {  t += " and B"; }
  void process()
     {  String s = “A";
        add(s);
     }
}
```

String & StringBuffer

- StringBuffers
  - are mutable
  - contain mutator methods
  - more efficient than String
  - fewer objects created
  - Example

```
void process()
{
        StringBuffer s = newStringBuffer(“house");
        s.append(“fly");
}
```

String & StringBuffer

- StringBuffer parameters
  - They can be updated in method.
  - No new object is created.

```
class AboutStringBuffer
{
        void change(StringBuffer t)
        {
                t.append(“fly"); // no new object
        }
        void process()
        {
          StringBuffer s = new StringBuffer(“butter");
          change(s);
        }
}
```

String & StringBuffer

- Create Strings
  - using a literal or using *new*
  - The creation with new is less efficient

```
void create()
{
        String s = “Amigo"; //One object is created.
        String t = new String(“My friend");
                                      // Two objects are created.
}
```

String & StringBuffer

- Create StringBuffers
  - Must use *new*
  - can create empty
  - can specify initial capacity
  - can specify initial character content

```
void create()
{
        StringBuffer sE = new StringBuffer();       //empty
        StringBuffer sC = new StringBuffer(32);   //32 chars
        StringBuffer sInit= new StringBuffer(“love");
}
```

String & StringBuffer

- String methods
  - Find them in API, package java.lang, class String
  - String class provides many methods
    - length
    - equals
    - compareTo
    - charAt
    - indexOf
    - lastIndexOf
    - subString
    - etc.

String & StringBuffer

- String methods: length()
  - returns number of characters in string

String myStr = "Hello";

int len = myStr.length(); len assigned 5

String & StringBuffer

- String methods: **equals()**
  - true if strings are equal (case sensitive)
  - false otherwise

if (a.equals(b)) …

- String methods: **equalsIgnoreCase()**
  - true if strings are equal (case insensitive)
  - false otherwise

String s =“Date”;

if (s.equalsIgnoreCase(“dAtE"))…

String & StringBuffer

- String methods: **substring()**
  - passed start and end indices
  - returns substring at specified indices
  - run time exception thrown on invalid index

```
String s = “0123456789";
String t = s.substring(2, 6);  //returns “2345"
```

String & StringBuffer

- StringBuffer methods:
  - **append**
  - **insert**
  - **delete, …**
- They are described in
  - API Specification
  - Package java.lang
  - Class StringBuffer

String & StringBuffer

- StringBuffer methods: **append()**
  - **adds to an existing StringBuffer**

    s.append(t); // makes s+t

- StringBuffer methods: **insert()**
  - **adds string at specified index**
  - **index must be in bounds**
  - **runtime exception thrown on invalid index**

StringBuffer s = new StringBuffer("0123456789");

String t = "abc";

s.insert(5, t); // sets s to be 01234abc56789

String & StringBuffer

- StringBuffer methods: `delete()`
  - **removes substring at specified indices**
  - **index must be in bounds**
  - **runtime exception thrown on invalid index**

StringBuffer s = new StringBuffer(“01abcde23”);

s.delete(2, 7); sets s to “0123"

String & StringBuffer

- common methods: **toString()**
  - **converting objects to Strings**
  - **Classes typically implement toString() method**
  - **returns string representation of object**
  - **toString() called automatically as needed**

```
Rational r = new Rational(3,4);
System.out.println(r);  //system look for r.toString()
```

  - **In class Rational you must build method toString()**