Strings

- Sequence of `char` characters.
- Usually implemented as a `(char[])` array.
- Provides array-like methods:
  - `int length()` returns string length.
  - `char charAt(int i)` returns ith character.
- Does not provide array-like assignment.
- (More later.)
String Reversing Program

• Repeatedly read a string from the keyboard.

• If the string is empty ("") then exit.

• Otherwise reverse the string and print it out.
private String reverse(String forward) {
    String backward = "";
    for (int i = 0; i < forward.length(); i++) {
        backward = forward.charAt(i) + backward;
    }
    return backward;
}

On each iteration, the \textit{i}th character is converted into a string of length one. Then the resulting string is appended to the beginning of the current backward string.

Strings are immutable. Once formed, they cannot be changed. Therefore each append (+) operation makes copies of the strings being appended in the course of forming the combined string.
Strings and String Buffers

• Strings:
  – Once created they cannot be changed.
  – Modifications require copying.
  – May be used to initialize `StringBuffer` objects.

• String Buffers:
  – May be modified.
  – May be used to initialize `String` objects.
private static String reverse(String forward) {
    StringBuffer backward = new StringBuffer();
    for (int i=forward.length()-1; i>=0; i--)
        backward.append(forward.charAt(i));
    return new String(backward);
}

StringBuffer objects are mutable: They may be modified after they are created. On each iteration, the (length-i-1)th character is appended to the end of the backward string. The StringBuffer append operation copies only the character being appended, not the StringBuffer to which it is being appended.
Command Line Arguments

- Java provides a mechanism to pass parameters to the `main` method of a class.
- Suppose Java program is invoked from a command line: “`java <Program> ...`”.
- Java will collect all of the text typed after `<Program>` and pass it to `main` as an array of strings.
public class CommandLineArgs {
{
    public static void main(String[] args) {
        { for (int i = 0; i < args.length; i++)
            System.out.println(args[i]);
        }
    }
}

> java CommandLineArgs larry curly moe
larry
curly
moe
LMB click on project. Press hold RMB to invoke project properties.
Reading the code:

```java
package cmdlineparms;

public class CmdLineParms {

    public static void main(String[] args) {
        System.out.println("0th parameter: " + args[0]);
        System.out.println("1st parameter: " + args[1]);
    }
}
```

Java passes command line arguments to the `main` method of an application via: `String[] args` (an array of strings).
Adding Command Line Parameters

```java
package argumentoperations;

public class ArgumentOperations {
    public static void main(String[] args) {
        System.out.println("Sum of arguments: " + args[0] + args[1]);
    }
}
```

What is wrong with this? What will Java print out?
ParseInteger Project

• Conversion of string to integer.
• Conversion of character to integer.
• Conversion of integer to boolean array.
• (Doing all the above explicitly with our own Java methods – even though we could get Java to do each conversion in one step.)
String to (Non-Negative) Integer

• Let sum be zero.

• For each string character, left to right:
  – Multiply sum by ten.
  – Convert character to an integer.
  – Add integer to sum.
private int stringToInteger(String text) {
    int integer = 0;
    for (int i = 0; i < text.length(); i++) {
        char digitChar = text.charAt(i);
        int digit = charToInteger(digitChar);
        integer = integer * 10 + digit;
    }
    return integer;
}

private int charToInteger(char c) {
    return c - '0';
}

A Java char (character) is an unsigned integer in the range 0 … 32767. Therefore we can treat it as an integer and do simple arithmetic. What are we assuming about Java’s encoding of characters '0' through '9' as unsigned integers?
private boolean[] integerToBitArray(int integer) {
    boolean[] bits = new boolean[32];
    for (int i = 0; i < 32; i++) {
        bits[i] = integer % 2 == 1 ? true : false;
        integer = integer / 2;
    }
    return bits;
}

We generate the bits in order from least to most significant. On each iteration, we extract the least significant bit using the mod % operator, and we truncate the integer using the integer division / operator.

Notice that the least significant bits are stored at the beginning of the boolean array. (Important later.)
private void printBits(boolean[] bits) {
    for (int i = 31; i >= 0; i--) {
        char bitChar = bits[i] ? '1' : '0';
        print(bitChar);
    }
    println();
}

We must print the most significant bits first – that’s the convention for printed representation of numbers. Therefore we must scan the boolean array in reverse order.
package typecharacter;
import java.io.*;
public class TypeCharacter {

    public static void main(String[] args) throws IOException {
        System.out.print("Type a character: ");
        int currentInt = System.in.read();
        char currentChar = (char) currentInt;
        System.out.println("You typed: " + currentChar);
    }
}

The read operation of the System.in stream returns an integer. The integer may be cast into a character.
package typeint1;
import java.io.*;

public class TypeInt1 {

    public static void main(String[] args) throws IOException {
        InputStreamReader inputStream = new InputStreamReader(System.in);
        BufferedReader inStream = new BufferedReader(inputStream);

        System.out.print("Type an integer: ");
        String currentLine = inStream.readLine();
        Integer currentInteger = Integer.valueOf(currentLine);
        int currentInt = currentInteger.intValue(); //This also works:
        //int currentInt = Integer.valueOf(currentLine);
        System.out.println("You typed: " + currentInt);
    }
}

The expression `Integer.valueOf(currentLine)` creates an `Integer` class object. The `intValue()` operation converts this into a primitive integer.
Integer currentInteger = Integer.valueOf(currentLine);

The `Integer.valueOf` method converts a string into an instance of the `Integer` class. What happens if the string does not represent an integer?

```java
int currentInt = currentInteger.intValue();
```

The `intValue` method of the `Integer` class converts an `Integer` object into a primitive `int` integer.

```java
int currentInt = Integer.valueOf(currentLine);
```

Here the two steps above are combined in a single statement.
Reading in a Single Integer

The expression `numberFormatter.parse(currentLine)` creates a `Number` class object. The `intValue()` operation converts this into a primitive integer, truncating any fractional part.
The expression `Double.valueOf(currentLine)` creates a `Double` class object. The `doubleValue()` operation converts this into a primitive double.
Reading in a Single Double

The expression `numberFormatter.parse(currentLine)` creates a `Number` class object. The `doubleValue()` operation converts this into a primitive double.
Wrapping the `System.in` Stream

- An `InputStreamReader` converts a `byte` stream into a `char` stream.

- A `BufferedReader`:
  - Provides useful operations (e.g. `readLine`) for reading an input stream.
  - Reads and stores large blocks of data for greater efficiency.
package inputecho;

import java.io.*;

public class InputEcho {

    public static void main(String[] args) throws IOException {
        InputStreamReader inputStream = new InputStreamReader(System.in);
        BufferedReader inStream = new BufferedReader(inputStream);
        String inputLine = null;
        do {
            System.out.print("<<< ");
            inputLine = inStream.readLine();
            System.out.println(">>> " + inputLine);
        } while (!inputLine.equals("")());
    }
}

Reads one line of input text at a time and prints it back out.
package typedoubles;
import java.io.*;
import java.text.*;
import java.util.StringTokenizer;

public class TypeDoubles {
    public static void main(String[] args) throws IOException, ParseException {
        InputStreamReader inputStream = new InputStreamReader(System.in);
        BufferedReader inStream = new BufferedReader(inputStream);
        NumberFormat numberFormatter = NumberFormat.getInstance();
        System.out.print("Type some doubles: ");
        String currentLine = inStream.readLine();
        StringTokenizer stringTokenizer = new StringTokenizer(currentLine);
        System.out.print("You typed:");
        while (stringTokenizer.hasMoreTokens()) {
            Number currentNumber = numberFormatter.parse(stringTokenizer.nextToken);
            double currentDouble = currentNumber.doubleValue();
            System.out.print(" "+currentDouble);
        }
        System.out.println();
    }
}

Reading in Several Doubles on One Line
StringTokenizer

• A `StringTokenizer` object is initialized with a `String` object.
• The tokenizer normally splits the string into parts separated by white space.
• The `nextToken` operation returns the next part as a `String` object.
• The `hasMoreTokens` operation tests whether any parts are left.
public class TypeString1 {

    private static final int EOF = -1;

    public static void main(String[] args) throws IOException {
        StringBuffer currentString = new StringBuffer();
        System.out.println("Type in a string. End with Control-D:");
        do {
            int currentInt = System.in.read();
            if (currentInt == EOF) {
                break;
            }
            char currentChar = (char) currentInt;
            currentString.append(currentChar);
        } while (true);
        System.out.println(currentString);
    }
}
package typestring2;
import java.io.*;

public class TypeString2 {
    public static void main(String[] args) throws IOException {
        InputStreamReader inputStreamReader = new InputStreamReader(System.in);
        BufferedReader bufferedReader = new BufferedReader(inputStreamReader);
        StringBuffer currentString = new StringBuffer();
        String currentLine;
        System.out.println("Type in a string. End with a blank line:");
        do {
            currentLine = bufferedReader.readLine();
            if (currentLine.length() == 0) {
                break;
            }
            currentString.append(currentLine);
            currentString.append('
');
        } while (true);
        System.out.println(currentString);
    }
}
Palindomer Program

• Repeatedly reads a string from the user.
• Removes punctuation and white space.
• Converts characters to lower case.
• Checks whether the result is a palindrome.
Palindromer Output

Enter a palindrome: radar
You got one!
Enter a palindrome: foobar
Sorry. Try again!
Enter a palindrome: Madam, I'm Adam!
You got one!
Enter a palindrome: A man, a plan, a canal: Panama.
You got one!
public void run() {
    while (true) {
        String candidate = readLine("Enter a palindrome: ");
        String clean = cleanUp(candidate);
        if (isPalindrome(clean)) {
            println("You got one!");
        } else {
            println("Sorry. Try again!");
        }
    }
}

public static void main(String[] args) {
    new Palindromer().start();
}
public static final String dirt = "~-!@#$%^&*()-+={}|[]\":;'<?,.\" ";

private boolean isClean(char c) { return dirt.indexOf(c) == -1; }

private String cleanUp(String candidate) {
    StringBuffer buff = new StringBuffer();
    for (int i = 0; i < candidate.length(); i++) {
        if (isClean(candidate.charAt(i))) {
            buff.append(candidate.charAt(i));
        }
    }
    return buff.toString().toLowerCase();
}

dirt.indexOf(c)
Returns the position of \texttt{c} in dirt or else \texttt{-1} if \texttt{c} does not appear.

buff.toString().toLowerCase();
Converts \texttt{buff} to a string and then converts the string to lower case.
Recursive Structure of Palindromes

private boolean isPalindrome(String candidate) {
    int length = candidate.length();
    if (length <= 1) {
        return true;
    } else if (candidate.charAt(0) != candidate.charAt(length - 1)) {
        return false;
    } else {
        return isPalindrome(candidate.substring(1, length - 1));
    }
}
Sub String Extraction

• The **String** class **substring** method takes two int parameters: **begin** and **end**.

• It returns a new **String** containing the characters in the original **String** from **begin** to **end−1**, in order.
Cryptogram Program

• Randomly selects one of several secret messages.
• Presents the user with an encrypted version of the message.
• User interactively tries to decode the message.
Cryptogram = NYBSDZNSSQFSNYBSYZN
   Guess = nt
N --> T
Cryptogram = NYBSDZNSSQFSNYBSYZN
   Guess = T T T T T
yh
Y --> H
Cryptogram = NYBSDZNSSQFSNYBSYZN
   Guess = TH T TH H T
be
B --> E
Cryptogram = NYBSDZNSSQFSNYBSYZN
   Guess = THE T THE H T
s
S --> dc
D --> C
Cryptogram = NYBSDZNSSQFSNYBSYZN
   Guess = THE C T THE H T
za
Z --> A
Cryptogram = NYBSDZNSSQFSNYBSYZN
   Guess = THE CAT THE HAT
qi
Q --> I
Cryptogram = NYBSDZNSSQFSNYBSYZN
   Guess = THE CAT I THE HAT
fn
F --> N
Cryptogram = NYBSDZNSSQFSNYBSYZN
   Guess = THE CAT IN THE HAT
You figured it out!