CS102

Introduction to
data structures, algorithms, and
object-oriented programming

DAY 4

Strings

Strings are sequences of characters. Methods we will use on Strings include length, toUpperCase, charAt, indexOf, substring (look these up in the Java API):

"abcdefg".length() returns 7

"tomorrow".toUpperCase() returns "TOMORROW"

String petString = "cats and dogs"; // creating a new string object

petString.charAt(6) returns 'n'

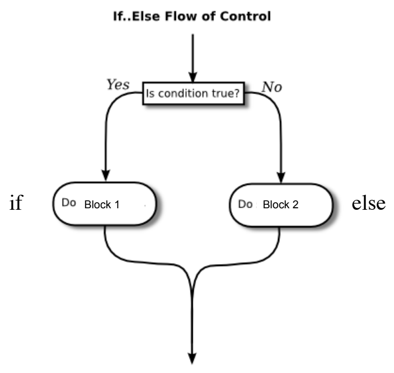
petString.indexOf('o') returns 10

petString.indexOf('X') returns -1

if Decision Statement – Eck 3.5

1. **if …else**: "either-or" type statement, each with its own block of code.
2. **if** alone with a block of code, only runs block if the expression is true, otherwise skips block.
3. **if**, **else if**, **else if**, …, **else**. Multi-way decision statement, each part with its own block of code.
4. **?:** Short form of if…else (either or)

if and else are like cond in Racket. Only one clause in the group is executed and the rest are ignored. The else at the end is like that in the cond, a default condition.



if Decision Statement

```
String line= javax.swing.JOptionPane.showInputDialog
             ("Please enter a line of text: ");
int count = line.length();
if (count > 1){
   System.out.println("Your input contains "+
                      count+" letters.\n");
}
else {
   System.out.println("Your input contains "+
                      count+" letter. \n");
}
```

?: operator—alternative to if else

```
String line= javax.swing.JOptionPane.showInputDialog
             ("Please enter a line of text: ");
int count = line.length();
System.out.println("Your input contains "+count+
       ((count>1) ? " letters.\n": " letter. \n"));
```

This code snippet first reads a line of text from the user and then prints the result. Either "letters" or "letter" is embedded in the String that is printed.

break, continue, and return

Java provides a general method for breaking out of the middle of any loop. It's called the **break** statement, which takes the form

break;

If you use a **break** statement inside a nested loop, it will only break out of innermost loop that contains the break, not out of the loop that contains the nested loop.

A **continue** statement tells the computer to skip the rest of the current iteration of the loop. However, instead of jumping out of the loop altogether, it jumps back to the beginning of the loop and continues with the next iteration.

A return statement exits the method and returns control to the line in which the method was called.

break

One place a break is used is in a while loop that expects a certain type of input:

```
int i = 0;
java.util.Scanner sc = new java.util.Scanner(System.in);
while ( true ) {
   System.out.println("Please enter a positive whole number");
   i = sc.nextInt();
   if ( i > 0) {
      break;      // correct input entered, go to line after
                  // end of while loop
    }
    // incorrect input entered, ask user for input again
    // by returning to the top of the while loop.
    System.out.println("Input must be positive.);
} // end of while loop
```

This loop will continue until the user enters a whole number greater than 0.

continue

This statement can be used in a loop to skip subsequent lines in loop and go back to the start of the loop:

```
int i = 0;
int sumEven = 0;
int count = 0;
java.util.Scanner sc = new java.util.Scanner(System.in);
while ( true ) {
   System.out.println("Please enter 5 positive whole numbers");
   System.out.println("and I will add the even numbers.");
   i = sc.nextInt();
   if ( i < 0) {
      System.out.println("Oops, that was a negative number.");
      continue;  // skip lines below if and go back to top of while
   }
   sumEven = sumEven + i; // i must be positive
   count++;
   if (count == 5) {
        break;   // need a way to break out of loop
   }
}
System.out.println("The sum of the even numbers is "+ sumEven);
```

Variations of the for loop § 3.4.1

Give three variations of for loops to print all the odd numbers between 1 and 21 :

```
for (int i = 1; i <=21; i+=2) {           /** Use a for loop to count 1, 3, ..., 21
   System.out.println(i);                   */
} // end for

for (int j = 1; j <= 21; j++) {           /** Use a for loop to count 1... 21, but
   if ( j % 2 ==1 ) {                      *   only print the numbers that are odd
     System.out.println(j);                */
   } // end if
} // end for                              /** Use a for loop to count k=1...10 and
                                           * and print the numbers 2k + 1.
for (int k=0; k<=10; k = k+1) {            */
   System.out.println(2*k + 1);
} // end for
```

for loops can also count down instead of up.

Enhanced for loop (foreach loops)

```
class EnhancedForDemo {

   public static void main(String[] args){

      int[] numbers = {1,2,3,4,5,6,7,8,9,10};
      int  sum = 0

      for (int  item :  numbers) {
         sum+= item;
         System.out.println("Sum is: " + sum);
      }
   }
}
```

foreach loops can be used on any collection of data.

Nested For Loops § 3.4.3

Control structures can contain other control structures. In particular, for loops are often nested.

```
for ( int rowNumber = 1; rowNumber <= 12; rowNumber++ ) {
   // for each row, process all columns
   for ( int N = 1; N <= 12; N++ ) {
      System.out.printf( "%4d", N * rowNumber );
      // print ints in 4-character columns; No newline
    }
    System.out.println(); // Add a newline
}
```

Nested loops

```
String str;  // Line of text entered by the user.
int count;   // Number of different letters found in str.
char letter; // A letter of the alphabet.
java.util.Scanner scan = new java.util.Scanner(System.in);

System.out.println("Please type in a line of text.");
str = scan.nextLine(); // call to nextLine method in Scanner class
str = str.toUpperCase(); // call on non-static method in object str
count = 0;  // initialize count
System.out.println("Your input contains the following letters:");
System.out.println();

for ( letter = 'A'; letter <= 'Z'; letter++ )
{
    for ( int i = 0; i < str.length(); i++ ) {
        if ( letter == str.charAt(i) ) {
            System.out.print(letter);
            System.out.print(' ');
            count++;
            break;
         }
     }
}
```

13

switch

A switch statement allows you to test the value of an expression x and to jump directly to some location within the switch statement, depending on the value of x.

The value of the expression listed in parentheses immediately to the right of the word switch can be one of the primitive integer types int, short, or byte. It can also be the primitive char type or it can be a String.

The expression **cannot** be a double or float value, nor can it be of object (reference) type.

14

switch example

```
switch ( N ) {// (Assume N is an integer variable or exp.)
   case 1:    // if N == 1
      System.out.println("The number is 1.");
      break;
   case 2:
   case 4:
   case 8:      // if N = 2, 4, or 8
      System.out.println("The number is 2, 4, or 8.");
      System.out.println("(That's a power of 2!)");
      break;
   case 3:
   case 6:
   case 9:      // if N = 3, 6, or 9
      System.out.println("The number is 3, 6, or 9.");
      System.out.println("(That's a multiple of 3!)");
      break;
   case 5:      // if N = 5
      System.out.println("The number is 5.");
      break;
   default:
      System.out.println("The number is 7 or is outside the");
      System.out.println(" range 1 to 9.");
}
```

15

arrays § 3.8

A data structure in which the items are arranged as a numbered sequence, so that each individual item can be referred to by its position number.

All the items in an array must be of the same type, and the numbering always starts at zero. An array is a list of variables, each accessible by the array name and position number of the variable.

An array is an object, so the process of creating one requires an instantiation with the keyword new.

16

arrays (cont.)

An array can be of any type and must first be declared:

```
String[] name;     // declaration of String array
int[] age;          // declaration of int array
boolean[] leftHanded; // declaration of boolean array
```

Then the array must be instantiated:

```
name = new String[1000];  // each with initial value null
age = new int[5];       // each with initial value 0
lefthanded  = new boolean[100]; // each is false initially
```

After instantiation, the specified number of boxes will be created in memory and reserved for that type.

17

arrays (cont.)

To put values into the array, you use the array name and position number to store a value at that position:

```
name[5] = "Penny";
```

The length of a array is stored with the array as a field name accessible as, for example   name.length // notice these are
                                    age.length   // not method calls

Having access to the length of every array allows them to be easily used with a for loop to go through each element:

```
// this for loop prints out all the elements in array called age
for (int i = 0; i < age.length; i++) {
   System.out.println( age[i] );
} // end for
```

18

2-dimensional arrays § 3.8.5

Declaration and instantiation example:

```
int[][] matrix; // declaration
matrix = new matrix[10][5]; // instantiation
```

This line would create a matrix with 10 rows and 5 columns, initially all 0's.

Often initialized or printed in nested for loops.

19

random numbers

The random method is a static member of the Math class. The call Math.random() produces a double between 0.0 and 1.0, inclusive. To use the Math.random() function to get a number between 1 and 10, you would use the following call:

```
int rNum = (int)(Math.random() * 10) + 1
```

The (int) operator truncates the real number to produce an int.

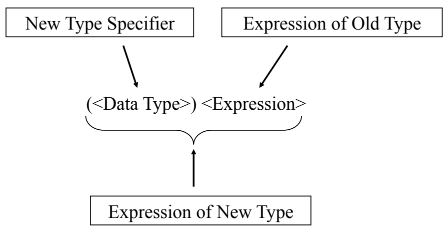
This type of operator is called a "cast".

20

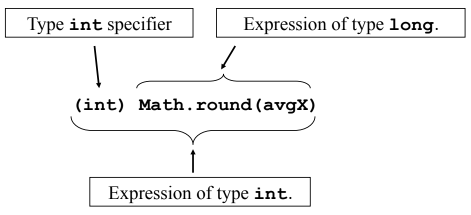
Type Conversion

- Changing a datum from one type into another.
- Explicit Conversion: Programmer uses a *cast* operation to perform the type conversion.
- Implicit Conversion: Compiler automatically inserts code to perform the type conversion.

Cast



Cast Example



Implicit Conversion from Narrow Types to Wider Types

$$\begin{tikzcd}
\texttt{byte} \arrow[r] & \texttt{short} \arrow[r] & \texttt{int} \arrow[r] \arrow[d] & \texttt{long} \arrow[d] \\
 & & \texttt{float} \arrow[r] & \texttt{double}
\end{tikzcd}$$

Algorithms – Eck 3.2

Step-by-step description of how to solve a problem.

Each line of human language must be broken down into a language solvable by a computer.

Developing a program from a human language form involves what is called *stepwise refinement*. That is, re-write each line into a form called *pseudocode* and then write it in a computer language.

25

Debugging Logical Errors

The hardest part of testing is to find bugs -- semantic errors that show up as incorrect behavior rather than as compilation errors.

Most programming environments come with a debugger, which is a program that can help you find errors by giving the value of different variables at a particular line in the code.

A more traditional approach to debugging is to insert debugging statements into your program. These are output statements that print out information about the state of the program.

26