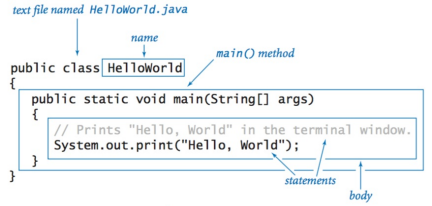
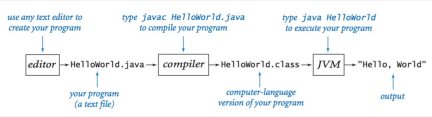
Basics

Hello, World



Editing, Compiling, and executing



Built in Datatypes

type	set of values	common operators	sample literal values
int	integers	+ - * / %	99 12 2147483647
double	floating-point numbers	+ - * /	3.14 2.5 6.022e23
boolean	boolean values	&& || !	true false
char	characters		'A' '1' '%' '\n'
String	sequences of characters	+	"AB" "Hello" "2.5"

Integers

values	integers between -2^{31} and $+2^{31}-1$					
typical literals	1234	99	0	1000000		
operations	sign	add	subtract	multiply	divide	remainder
operators	+ -	+	-	*	/	%

Integer Operations

expression	value	comment
99	99	integer literal
+99	99	positive sign
-99	-99	negative sign
5 + 3	8	addition
5 - 3	2	subtraction
5 * 3	15	multiplication
5 / 3	1	no fractional part
5 % 3	2	remainder
1 / 0		run-time error
3 * 5 - 2	13	* has precedence
3 + 5 / 2	5	/ has precedence
3 - 5 - 2	-4	left associative
(3 - 5) - 2	-4	better style
3 - (5 - 2)	0	unambiguous

Floating Point Numbers

values	real numbers (specified by IEEE 754 standard)			
typical literals	3.14159	6.022e23	2.0	1.4142135623730951
operations	*add*	*subtract*	*multiply*	*divide*
operators	+	-	*	/

Floating Point Numbers Operations

expression	*value*
3.141 + 2.0	5.141
3.141 - 2.0	1.111
3.141 / 2.0	1.5705
5.0 / 3.0	1.6666666666666667
10.0 % 3.141	0.577
1.0 / 0.0	Infinity
Math.sqrt(2.0)	1.4142135623730951
Math.sqrt(-1.0)	NaN

String Data Type

`public class String`

	String(String s)	*create a string with the same value as* `s`
	String(char[] a)	*create a string that represents the same sequence of characters as in* `a[]`
int	length()	*number of characters*
char	charAt(int i)	*the character at index* `i`
String	substring(int i, int j)	*characters at indices* `i` *through* `(j-1)`
boolean	contains(String substring)	*does this string contain* `substring`*?*
boolean	startsWith(String prefix)	*does this string start with* `prefix`*?*
boolean	endsWith(String postfix)	*does this string end with* `postfix`*?*
int	indexOf(String pattern)	*index of first occurrence of* `pattern`
int	indexOf(String pattern, int i)	*index of first occurrence of* `pattern` *after* `i`
String	concat(String t)	*this string, with* `t` *appended*
int	compareTo(String t)	*string comparison*
String	toLowerCase()	*this string, with lowercase letters*
String	toUpperCase()	*this string, with uppercase letters*
String	replace(String a, String b)	*this string, with* `a`*s replaced by* `b`*s*
String	trim()	*this string, with leading and trailing whitespace removed*
boolean	matches(String regexp)	*is this string matched by the regular expression?*
String[]	split(String delimiter)	*strings between occurrences of* `delimiter`
boolean	equals(Object t)	*is this string's value the same as* `t`*'s?*
int	hashCode()	*an integer hash code*

String operations

```
String a = new String("now is");
String b = new String("the time");
String c = new String(" the");
```

instance method call	*return type*	*return value*
a.length()	int	6
a.charAt(4)	char	'i'
a.substring(2, 5)	String	"w i"
b.startsWith("the")	boolean	true
a.indexOf("is")	int	4
a.concat(c)	String	"now is the"
b.replace("t", "T")	String	"The Time"
a.split(" ")	String[]	{ "now", "is" }
b.equals(c)	boolean	false

Commands to Split a String Into Separate Parts

```
Scanner scan = new Scanner(System.in);
               Sys tem.ou t.p rin tln ("What is your date of birth? (Format: dd mm yyyy) ");
               String dateBi rthday = scan.n ext Line();
               Str ing[] parts = dateBi rth day.sp lit (" ");
              // Storing users birthday data
              int day = Intege r.p ars eIn t(p art s[0]);
```

Commands to Split a String Into Separate Parts (cont)

```
>       int month = Integer.parseInt(parts[1]);
      int year = Integer.parseInt(parts[2]);
      // Obtaining today's date:
      int currentDay = LocalDate.now().getDayOfMonth();
      int currentMonth = LocalDate.now().getMonthValue();
      int currentYear = LocalDate.now().getYear();
```

Math Library

```
public class Math
```

`double abs(double a)`	*absolute value of a*
`double max(double a, double b)`	*maximum of a and b*
`double min(double a, double b)`	*minimum of a and b*
`double sin(double theta)`	*sine of theta*
`double cos(double theta)`	*cosine of theta*
`double tan(double theta)`	*tangent of theta*
`double toRadians(double degrees)`	*convert angle from degrees to radians*
`double toDegrees(double radians)`	*convert angle from radians to degrees*
`double exp(double a)`	*exponential (e^a)*
`double log(double a)`	*natural log ($\log_e a$, or $\ln a$)*
`double pow(double a, double b)`	*raise a to the bth power (a^b)*
`long round(double a)`	*round a to the nearest integer*
`double random()`	*random number in [0, 1)*
`double sqrt(double a)`	*square root of a*
`double E`	*value of e (constant)*
`double PI`	*value of π (constant)*

Type Conversions

expression	*expression type*	*expression value*
`(1 + 2 + 3 + 4) / 4.0`	`double`	`2.5`
`Math.sqrt(4)`	`double`	`2.0`
`"1234" + 99`	`String`	`"123499"`
`11 * 0.25`	`double`	`2.75`
`(int) 11 * 0.25`	`double`	`2.75`
`11 * (int) 0.25`	`int`	`0`
`(int) (11 * 0.25)`	`int`	`2`
`(int) 2.71828`	`int`	`2`
`Math.round(2.71828)`	`long`	`3`
`(int) Math.round(2.71828)`	`int`	`3`
`Integer.parseInt("1234")`	`int`	`1234`

Parsing Command-Line Arguments

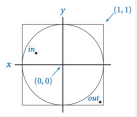
`int Integer.parseInt(String s)`	*convert s to an int value*
`double Double.parseDouble(String s)`	*convert s to a double value*
`long Long.parseLong(String s)`	*convert s to a long value*

Swtich statement

```
switch (day) {
   case 0: System.out.println("Sun"); break;
   case 1: System.out.println("Mon"); break;
   case 2: System.out.println("Tue"); break;
   case 3: System.out.println("Wed"); break;
   case 4: System.out.println("Thu"); break;
   case 5: System.out.println("Fri"); break;
   case 6: System.out.println("Sat"); break;
}
```

Do-While Loop

```
do
{  // Scale x and y to be random in (-1, 1).
   x = 2.0*Math.random() - 1.0;
   y = 2.0*Math.random() - 1.0;
} while (Math.sqrt(x*x + y*y) > 1.0);
```



Example Functions

absolute value of an int value

```
public static int abs(int x)
{
   if (x < 0) return -x;
   else       return  x;
}
```

absolute value of a double value

```
public static double abs(double x)
{
   if (x < 0.0) return -x;
   else         return  x;
}
```

primality test

```
public static boolean isPrime(int n)
{
   if (n < 2) return false;
   for (int i = 2; i <= n/i; i++)
      if (n % i == 0) return false;
   return true;
}
```

hypotenuse of a right triangle

```
public static double hypotenuse(double a, double b)
{  return Math.sqrt(a*a + b*b);  }
```

harmonic number

```
public static double harmonic(int n)
{
   double sum = 0.0;
   for (int i = 1; i <= n; i++)
      sum += 1.0 / i;
   return sum;
}
```

uniform random integer in [0, n)

```
public static int uniform(int n)
{  return (int) (Math.random() * n);  }
```

draw a triangle

```
public static void drawTriangle(double x0, double y0,
                                double x1, double y1,
                                double x2, double y2 )
{
   StdDraw.line(x0, y0, x1, y1);
   StdDraw.line(x1, y1, x2, y2);
   StdDraw.line(x2, y2, x0, y0);
}
```

Typical Array Problems

create an array with random values

```
double[] a = new double[n];
for (int i = 0; i < n; i++)
   a[i] = Math.random();
```

print the array values, one per line

```
for (int i = 0; i < n; i++)
   System.out.println(a[i]);
```

find the maximum of the array values

```
double max = Double.NEGATIVE_INFINITY;
for (int i = 0; i < n; i++)
   if (a[i] > max) max = a[i];
```

compute the average of the array values

```
double sum = 0.0;
for (int i = 0; i < n; i++)
   sum += a[i];
double average = sum / n;
```

reverse the values within an array

```
for (int i = 0; i < n/2; i++)
{
   double temp = a[i];
   a[i] = a[n-1-i];
   a[n-i-1] = temp;
}
```

copy sequence of values to another array

```
double[] b = new double[n];
for (int i = 0; i < n; i++)
   b[i] = a[i];
```

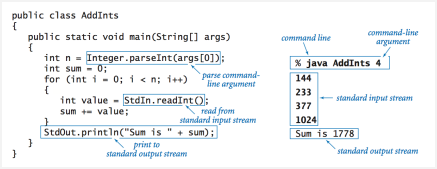
Formatted Output

%d	Integer in base 10 (“decimal”)	12345
%,d	Integer with comma separators	12,345
%08d	Padded with zeros, at least 8 digits wide	00012345
%f	Floating-point number	6.789000
%.2f	Rounded to 2 decimal places	6.79
%s	String of characters	"Hello"
%x	Integer in base 16 (“hexadecimal”)	bc614e

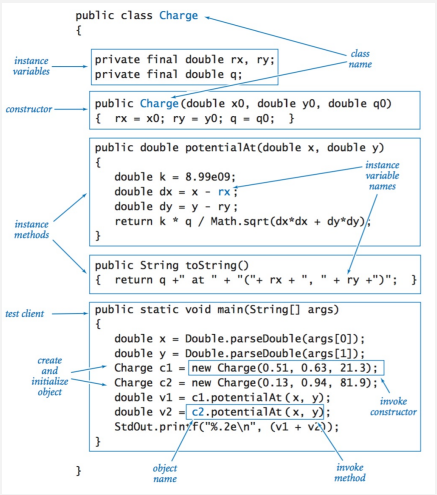
Table 3.1: Example format specifiers

Memory Usage of Data Types

Command Line Input from File



Classes Overview



Method to Read a Matrix From a File

```
public static double[][] readMatrix(String filename)
        throws java.i o.F ile Not Fou ndE xce ption{
       File file = new File(f ile name);
        Scanner input = new Scanner (file);

       int rows = input.n ex tInt();
```

Method to Read a Matrix From a File (cont)

```
>    int columns = input.nextInt();

   double[][] matrix = new double[rows][columns];

   for (int i=0;i<rows;i++){
    for (int j=0;j<columns;j++){
     matrix [i] [j]= input.nextDouble();

    }
   }

   input.close();
   return matrix;
 }
```

Two first Integers in a file defines the size of a matrix n x m.

Tutorial Content

Random Arrays

```
// returns an array of length n containing random integer values
      public static int[] random Int Arr ay(int n){
            Random random = new Random();
            int[] array = new int[n];
            for(int i=0; i < n; i++){
                  arr ay[i] = random.ne xtI nt();
           }
            return array;
     }

     // returns an array of length n containing random integer values
     // in the interval [lb,ub]
      public static int[] random Int Arr ay(int n, int lb, int ub){
            Random random = new Random();
            int[] array = new int[n];
            for(int i=0; i < n; i++){
                  arr ay[i] = random.ne xtI nt( ub- lb+1) + lb;
           }
```

Random Arrays (cont)

```
>       return array;
  }

  // returns an array of length n containing random double values
  public static double[] randomDoubleArray(int n){
    Random random = new Random();
    double[] array = new double[n];
    for(int i=0; i < n; i++){
      array[i] = random.nextDouble();
    }
    return array;
  }

  // returns an array of length n containing random double values
  // in the interval [lb,ub]
  public static double[] randomDoubleArray(int n, int lb, int ub){
    Random random = new Random();
    double[] array = new double[n];
    for(int i=0; i < n; i++){
      array[i] = random.nextDouble(ub-lb+1) + lb;
    }
    return array;
  }
```

Read Double Array.

```
public static double[] readArray(String filename)
         throws java.i o.F ile Not Fou ndE xce ption{
        File file = new File(f ile name);
         Scanner input = new Scanner (file);

        int n = input.n ex tInt();

         dou ble[] double Array = new double[n];

        for (int i=0;i< n;i++){
             dou ble Array [i] = input.n ex tDo uble();
        }
```

Read Double Array. (cont)

```
>    input.close();

   return doubleArray;
 }
```

To read other type array, we substitute the "doubles" with int or int[] or nextInt()

Digit Sum Method

```
public static int totalDigitSum(int n) {
                int number OfD igits = (Strin g.v alu eOf (n) ).l eng th();
                int sum = 0;
                 Sys tem.ou t.p rin t("The total digit sum");
                 whi le( num ber OfD igits != 0) {
                        sum = sum + (n%10);
                        n = n/10;
                         num ber OfD igi ts--;
                }
                 return sum;
        }
```

Print Prime Factors

```
public static void printPrimeFactors(int n) {
                if (n <= 1) {
                         Sys tem.ou t.p rin t("No prime factor s");
                         return;
                }
                for (int i = 2; i <= n; i++) {
                         while (n % i == 0) {
                                  Sys tem.ou t.p rint(i + " ");
                                 n /= i;
                        }
                }
        }
```

Method to print the prime factors of a given number

Method to compute Factorials (n!)

```
public static int[] computeFactorials(int n) {
                int[] factorials = new int[n];
                fac tor ials[0] = 1;

               for (int i = 1; i < n; i++) {
                        fac tor ials[i] = factor ials[i - 1] * (i + 1);
               }

                return factor ials;
       }
```

Method Simulating Dice Rolling N times

```
public static double expectedValue (int numberOfTrials){

            double sum = 0;

            for(int i = 1; i <= number OfT rials; i++){
                 sum = sum + (int) (Math.r an dom() * 6) +1;
           }
            return sum/nu mbe rOf Trials;
     }
```

Program that computes the expected value of rolling a six sided dice. Input parameter is the number of trials

Method for Standard Deviation

```
// Method that computes standard deviation of an array
        public static double getSta nda rdD evi ati on( dou ble[] array) {
                double sum = 0;
                double mean = getMea n(a rray);
                for (double i : array) {
                        double temp = (i - mean) * (i - mean);
                       sum += temp;
               }
                double standa rdD evi ation = sum/ar ray.le ngth;
                sta nda rdD evi ation = Math.s qrt (st and ard Dev iat ion);
                return standa rdD evi ation;
       }
```

Greatest Common Divisor & Least Common Multiple

```
// GCD
public static int getGcd(int n, int m) {
                whi le(m%n != 0) {
                        int temp = m%n;
                        m = n;
                        n = temp;
                }
                 return n;
        }
// LCM
public static int lcm(int a, int b) {
                int gcd = gcd(a, b);
                 return (a / gcd) * b;
        }
```

Determening the Season

```
/* Program that asks the user for a date and returns in which season this date falls.
// method calcul ating determ ining the season, given the month and day
         public int getSea son(int day, int month) {
                // Formula to translate everything on one 'scale'
                int combin edValue = month*100 + day;
                if (combi ned Value >= 321 & combin edValue <= 620) {
                          thi s.s eason = 1;
                }
                else if (combi ned Value >= 621 & combin edValue <= 920) {
                          thi s.s eason = 2;
                }
                else if (combi ned Value >= 921 & combin edValue <= 1220) {
                          thi s.s eason = 3;
                }
                else {
                          thi s.s eason = 0;
```

Determening the Season (cont)

```
>     }
      return this.season;
  }
```

Sorting Algorithms

Commands for Random Numbers

```
double random1 = Math.random();

             Sys tem.ou t.p rin tln ("A random value between 0 and 1: " + random1);

             double random2 = Math.r and om( )*100;

             Sys tem.ou t.p rin tln ("A random double value between 0 and 100: " + random2);

            int random3 = (int) (Math.r an dom ()* 100);

             Sys tem.ou t.p rin tln ("A random integer value between 0 and 99: " + random3);
```

Binary Search Method

```
public static boolean find(int[] array, int number){

            int low = 0;
            int high = array.l en gth-1;

             while (low+1 < high) {
                  int mid = (high + low) /2;

                  if (array [mid] == number){
                         return true;
                  }
                  else if (array [mid] < number){
                        low = mid;
                  }
                  else {
                        high = mid;
                  }
            }

             return ((arra y[low] == number) || (array [high] == number));
      }
```

Random Sorted Int Array

```
public static int[] randomSortedIntArray(int n, int startValue, int maxIncrement){
            Random random = new Random();
            int[] array = new int[n];
            arr ay[0] = startV alue;
            for(int i=1; i < n; i++){
                  arr ay[i] = array[i-1] + random.ne xtI nt( max Inc rem ent);
           }
            return array;
     }
```

Random Sorted Int Array (Basis for Binary Search)

Insertion Sort Algorithm

```
public static void sort(int[] a) {
       int n = a.length; // Get the length of the input
                                                                  array 'a'.
       for (int i = 1; i < n; i++) {
               for (int j = i; j > 0; j--) {
                       // This loop iterates from the current
                                  pos ition 'i' towards the beginning
                                 of the array.
                       if (a[j-1] > a[j]) {
                               // If the element at the previous
                                        pos ition is greater than the
                                          current element,
                               // we need to swap them to sort
                                       the array in ascending order.
                                swap(a, j-1, j);
                       } else {
                               // If the element at the previous
                                  pos ition is not greater than the
                                  current element,
                               // it means the array is sorted
```

Insertion Sort Algorithm (cont)

```
>             up to this point, so we break out of the
              loop.
              break;
            }
        }
    }
}
public static void swap(int[] arr, int index1, int index2) {
    // This function swaps two elements
       in the array 'arr' at the given indices 'index1'
         and 'index2'.
    int temp = arr[index1];
    arr[index1] = arr[index2];
    arr[index2] = temp;
}
```

Random Array in Interval

```
public static int[] randomArray(int n){
      int[] array = new int[n];

      for(int i=0; i<n ;i++){
        array [i] = (int) ( Math.r and om( )*1000 - 500);
     }
      return array;
   }
```

random array length n, interval int between [-500;500]

Insertion Sort of an Object w.r.t. String/Int

```
// Insertion Sort Algorithm based on First Name
      public static void sortFi rst Nam e(P ers onNew[] person List) {
            for(int i = 1; i < person Lis t.l ength; i++) {
                  Per sonNew curren tPerson = person Lis t[i];
                int j = i - 1;
```

```
>       boolean flag = true;
      // We will compare current person's name with other ones
      while(j >= 0 && flag) {
        String currentName = currentPerson.getFirstName();
        String testName = personList[j].getFirstName();
        // Checking which name is shorter:
        int minimum = Math.min(currentName.length(), testName.length());
        int k = 0;
        // Now we will check which name is "smaller" based on ascii:
        while(k < minimum) {
          char letterCurrent = currentName.charAt(k);
          char letterTest = testName.charAt(k);
          // Current Name is larger than other
          if(letterCurrent < letterTest) {
            personList[j+1] = personList[j];
            j--;
            break;
          }
          // Current Name is already smaller
          else if(letterTest < letterCurrent) {
            flag = false;
            break;
          }
          // They have the same characters, thus we continue going through each string
          else {
            k++;
          }
          // We reach the end of smaller name, meaning that they have the same name:
          if(k == minimum) {
            personList[j+1] = personList[j];
            j--;
```

```
>          }
        }
      }
      // Inserting in its correct position.
      personList[j+1] = currentPerson;
    }
  }
// Insertion Sort Algorithm based on Height
  public static void sortHeight(PersonNew[] personList) {
    for(int i = 1; i < personList.length; i++) {
      PersonNew currentPerson = personList[i];
      int j = i-1;
      while((j >= 0) && currentPerson.isTaller(personList[j])) {
        personList[j+1] = personList[j];
        j--;
      }
      personList[j+1] = currentPerson;
    }
  }
```

Objects

Constructor Method for Point / Line

```
// constructor for a line through points p and q
// sets slope and intercept to Double.NaN for vertical lines
        public Line(Point p, Point q) {
               // vertical line:
               if (p.getX() == q.getX()) {
                        thi s.i nte rcept = Double.NaN;
                        thi s.slope = Double.NaN;
               }
               // otherwise we compute the slope (y2 - y1) / (x2 - x1)
```

Constructor Method for Point / Line (cont)

```
>       // and intercept from y = mx + c, using one of the points
      else {
         this.slope = (q.getY() - p.getX()) / (q.getX() - p.getX());
         this.intercept = p.getY() - this.slope * p.getX();
      }
   }
```

This Constructor computes the equation of the line, given two points

Few Methods for Object Date

```
// method that returns true if this date is before the date given by the argument
        public boolean isBefo re(Date date) {
               if (this.year <= date.year) {
                        if( thi s.month <= date.m onth) {
                                if( thi s.day <= date.day) {
                                        return true;
                                }
                                return false;
                        }
                        return false;
               }
                return false;
       }
       // method that increments this date by one day
        public void increm ent() {
               // Array storing days per month
                int[] daysPe rMonth = {0, 31, 28, 31, 30, 31, 30, 31, 31, 30, 31, 30, 31};
               // Figure out if its leap year, to change daysPe rMo nth[2] = 29.
                if( isL eap Year()) {
                        day sPe rMo nth[2] = 29;
               }
               // Check if its last day of the month
                if( thi s.day == daysPe rMo nth [th is.m onth]) {
```

Few Methods for Object Date (cont)

```
>           this.day = 1;
          // Check if its new year:
          if(this.month == 12) {
             this.month = 1;
             year++;
          }
          else {
             month++;
          }
       }
       else {
          day++;
       }
    }
    // method that returns true if the year of this date is a leap year
    private boolean isLeapYear() {
       return (this.year % 4 == 0 && this.year % 100 != 0) || (this.year % 400 == 0);
}
```

We store dd/mm/yyyy in the Object Date. Constructor method is not shown here.

Methods For Rational Numbers (Object Rational)

```
// add rational f2 to this rational and return the result.
         public Rational add(Ra tional f2) {
                  if( thi s.d eno minator == f2.get Den omi nat or()) {
                        int newNum erator = this.n ume rator + f2.get Den omi nat or();
                         return new Ration al( new Num erator, this.d eno min ator);
                  }
                  else {
                        // we find the new denomi nator:
                        int newDen omi nator = findLC M(t his.de nom inator, f2.get Den omi nat or());
                        int newNum erator = (this.(newDeenumerator / this.d eno min ator)) +
(f2.ge tNu mer ator()(newDe nom inator / f2.get Den omi nat or()));
                         return new Ration al( new Num erator, newDen omi nator);
```

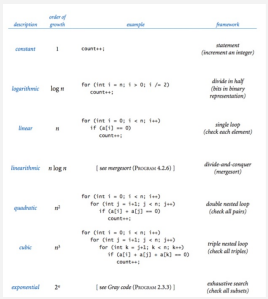
Methods For Rational Numbers (Object Rational) (cont)

```
>      }
   }
// compute the reciprocal of this Rational and return the result
   public Rational reciprocal() {
      return new Rational(this.denominator, this.numerator);
   }
// divide this Rational by Rational f2 and return the result
   public Rational divide(Rational f2) {
      Rational reciprocalf2 = f2.reciprocal();
      int newNum = this.numerator * reciprocalf2.getNumerator();
      int newDen = this.denominator * reciprocalf2.getDenominator();
      return new Rational(newNum, newDen);
   }
```

Reciprocal: x/y becomes y/x.

Runtime Exercises

Run Times



Knapsack Problem

```
public static int knapsackAlgorithm(int n, int[] profits, int[] volumes, int V) {
       // Create a 2D array 'P' to store the maximum profit for different combin ations
 of items and volumes.
        int[][] P = new int[n + 1][V + 1];
       // Initialize the first row (no items) with zeros, as no profit can be gained.
       for (int v = 0; v <= V; v++) {
```

Knapsack Problem (cont)

```
>       P[0][v] = 0;
    }
    // Loop through the items (i) and the available volume (v).
    for (int i = 1; i <= n; i++) {
        for (int v = 0; v <= V; v++) {
            // Check if the volume of the current item can fit in the
available volume (v).
            if (volumes[i - 1] <= v) {
                // If it can fit, calculate the maximum profit for two options:
                // 1. Including the current item and subtracting its volume
from the available volume.
                // 2. Excluding the current item.
                P[i][v] = Math.max(profits[i - 1] + P[i - 1]
[v - volumes[i - 1]], P[i - 1][v]);
            } else {
                // If the current item cannot fit, take the maximum profit
from the previous row.
                P[i][v] = P[i - 1][v];
            }
        }
    }
    // The final element of the 'P' array (bottom-right) contains the
maximum profit achievable with the given constraints.
    return P[n][V];
}
public static int[] generateRandomArray(int size, int min, int max) {
    // Create an array of the specified size.
    int[] array = new int[size];
    Random rand = new Random();

    // Fill the array with random integers in the range [min, max].
    for (int i = 0; i < size; i++) {
```

Knapsack Problem (cont)

```
>       array[i] = rand.nextInt(max - min + 1) + min;
   }

   // Return the generated array.
   return array;
}
```

Magic Square Problem

```
public static boolean isMagic(int[][] matrix) {
                return (hasUn iqu eNu mbe rs( matrix) && hasEqu alS ums (ma trix));
       }
       // Method that checks if all rows, columns and
two diagonals (right & left) have equal sums:
        public static boolean hasEqu alS ums (in t[][] matrix) {
               // Variable to store target sum:
               int targetSum = 0;
               // Target sum = sum of first row
                for(int i = 0; i < matrix.le ngth; i++) {
                        tar getSum += matrix [0][i];
               }
               int leftDi agonal = 0;
               int rightD iagonal = 0;
                for(int j = 0; j < matrix.le ngth; j++) {
                       int sumColumn = 0;
                       int sumRow = 0;
                        for(int k = 0; k < matrix [0].le ngth; k++) {
                                sumRow += matrix [j][k];
                                sum Column += matrix [k][j];
                               // Check if we are on the left diagonal:
                               if(j == k) {
                                        lef tDi agonal += matrix [j][k];
                               }
                               // Check if we are on the right diagonal:
```

Magic Square Problem (cont)

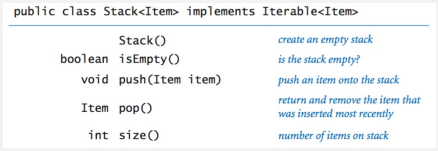
```
>                 if(j+k == matrix.length - 1) {
                    rightDiagonal += matrix[j][k];
                }
            }
            // Check if the sum is equal to our target one:
            if (sumRow != targetSum || sumColumn != targetSum) {
                return false;
            }
        }
        // Check if diagonal sums are = target
        if(rightDiagonal != targetSum || leftDiagonal != targetSum) {
            return false;
        }
        return true;
    }
    // Method that checks whether matrix contains
only unique numbers from 1 to n*n.
    public static boolean hasUniqueNumbers(int[][] matrix) {
        /*
         We know that there will be numbers from 1 to n*n.
         We create boolean array of size n*n which
will help to flag not unique numbers.
         Default array element value = false.
        */
        int n = matrix.length;
        boolean[] seenNumber = new boolean[n*n];
        int arrayIndex = 0;
        for(int i = 0; i < matrix.length; i++) {
            for(int j = 0; j < matrix[0].length; j++) {
                int number = matrix[i][j];
                // We check whether the number has
```

Magic Square Problem (cont)

```
> already been flagged && whether 1 <= number <= n*n
            if(!seenNumber[number - 1] && number >= 1 && number <= (n*n)) {
               // Number is being flagged:
               seenNumber[number - 1] = true;
            }
            else {
               return false;
            }
         }
      }
      return true;
   }
  public static int [][] rotate(int[][] matrix){
    int n = matrix.length;
    int m = matrix[0].length;
    // Create a new matrix to store the rotated elements
    int[][] transpose = new int[n][m];
    for(int i=0;i<n;i++){
      for(int j=0;j<n;j++){
        transpose[i][j]=matrix[j][i];
      }
    }
    int[][] rotated = new int[n][m];
    int x = n-1;
    for (int k = 0; k < n; k++) {
      rotated[k] = transpose[x];
      x--;
    }
    return rotated;
  }
```

Data Structures

Stack Data Type



public class Stack<Item> implements Iterable<Item>	
Stack()	*create an empty stack*
boolean isEmpty()	*is the stack empty?*
void push(Item item)	*push an item onto the stack*
Item pop()	*return and remove the item that was inserted most recently*
int size()	*number of items on stack*

Intger Stack Class

```
import java.util.*;
public class Intege rStack {
        private int size; // size of the stack
        private Node first; // top of stack
       // helper linked list class
        private class Node {
                private int value;
                private Node next;
       }
       // Constr uctor for an empty stack
        public Intege rSt ack() {
                thi s.first = null;
                thi s.size = 0;
       }
       // Returns true if this stack is empty.
        public boolean isEmpty() {
                return first == null;
       }
       // returns the size of this stack
        public int size() {
                return this.size;
       }
       // adds number to the top of the stack
        public void push(int number) {
               Node oldfirst = first;
```

Intger Stack Class (cont)

```
>       first = new Node();
        first.value = number;
        first.next = oldfirst;
        size++;
    }
    public String toString() {
        IntegerStack aux = new IntegerStack();
        String stackString = new String();
        while (!this.isEmpty()) {
            int x = this.pop();
            stackString += x + " ";
            aux.push(x);
        }
        while (!aux.isEmpty()) {
            this.push(aux.pop());
        }
        stackString += "\n";
        return stackString;
    }
    /**
     * Removes and returns the item most recently added to this stack.
     */
    public int pop() {
        // if (isEmpty()) throw new NoSuchElementException("Stack underflow");
        int number = first.value; // save item to return
        first = first.next; // delete first node
        size--;
        return number; // return the saved item
    }
    public void absoluteValue() {
        IntegerStack aux = new IntegerStack();
```

Intger Stack Class (cont)

```
>       while (!this.isEmpty()) {
          aux.push(Math.abs(this.pop()));
        }
        // put auxiliary stack back on both stacks
        while (!aux.isEmpty()) {
          this.push(aux.pop());
        }
    }
    // Method that checks if two stacks are exactly the same
    public boolean isEqual(IntegerStack secondStack) {
        IntegerStack aux = new IntegerStack();
        boolean flag = true;
        // Stacks are of a different size.
        if(this.size != secondStack.size) {
          flag = false;
          return flag;
        }
        // Checking whether the two are the same
        while(!this.isEmpty() && !secondStack.isEmpty()) {
          int thisNumber = this.pop();
          int secondNumber = secondStack.pop();
          if(!(thisNumber == secondNumber)) {
            // Immediately push back the numbers on top:
            this.push(thisNumber);
            secondStack.push(secondNumber);
            flag = false;
            break;
          }
          aux.push(thisNumber);
        }
        // Restoring both stacks:
```

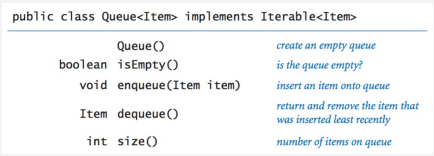
Intger Stack Class (cont)

```
>         while (!aux.isEmpty()) {
           int number = aux.pop();
           this.push(number);
           secondStack.push(number);
         }
         return flag;
    }
    // Method that reverses the order of the stack:
    public void reverse() {
        // empty stack:
        if(this.isEmpty()) {
            System.out.println("Empty stack was provided.");
            return;
        }
        Queue<Integer> tempQ = new Queue<Integer>();
        // Putting all elements from stack to queue data structure
        while(!this.isEmpty()) {
            tempQ.enqueue(this.pop());
        }
        // Putting everything form queue to stack, to get reverse order
        while(!tempQ.isEmpty()) {
            this.push(tempQ.dequeue());
        }
    }
// Fills stack with -1 and 1 randomly
    public void fillStackRandom() {
        Random randomNumber = new Random();
        for(int i = 0; i < 100; i++) {
            int number = randomNumber.nextInt(2) == 0 ? 1 : -1;
            this.push(number);
        }
```

Intger Stack Class (cont)

```
>  }
}
```

Queue Data Type



public class Queue<Item> implements Iterable<Item>

	Queue()	*create an empty queue*
boolean	isEmpty()	*is the queue empty?*
void	enqueue(Item item)	*insert an item onto queue*
Item	dequeue()	*return and remove the item that was inserted least recently*
int	size()	*number of items on queue*

Queue Class

```
import java.util.Iterator;
import java.u til.No Suc hEl eme ntE xce ption;
public class Queue< Ite m> implements Iterab le< Ite m> {
        private int n; // number of elements on queue
        private Node first; // beginning of queue
        private Node last; // end of queue
       // helper linked list class
        private class Node {
                private Item item;
                private Node next;
       }
// Initia lizes an empty queue.
        public Queue() {
                first = null;
               last = null;
               n = 0;
       }
// Returns true if this queue is empty.
        public boolean isEmpty() {
                return first == null;
       }
// Returns the number of items in this queue.
        public int length() {
```

Queue Class (cont)

```
>         return n;
    }
// Returns the item least recently added to this queue.
    public Item peek() {
        if (isEmpty()) throw new NoSuchElementException("Queue underflow");
        return first.item;
    }
// Add the item to the queue.
public void enqueue(Item item) {
        Node oldlast = last;
        last = new Node();
        last.item = item;
        last.next = null;
        if (isEmpty()) first = last;
        else oldlast.next = last;
        n++;
    }
// Removes and returns the item on this queue that was least recently added.
    public Item dequeue() {
        if (isEmpty()) throw new NoSuchElementException("Queue underflow");
        Item item = first.item;
        first = first.next;
        n--;
        if (isEmpty()) last = null; // to avoid loitering
        return item;
    }
// Returns a string representation of this queue.
    public String toString() {
        StringBuilder s = new StringBuilder();
        for (Item item : this) {
            s.append(item);
```

Queue Class (cont)

```
>            s.append(' ');
        }
        return s.toString();
    }
    /*
     method to remove all negative integers from a queue.
     it checks the last element of the queue, if its not negative
     element goes to the back of the queue,
    */
    public void removeNegatives() {
        int sizeQ = this.size();
        for(int i = 0; i < sizeQ; i++) {
            Item lastElement = this.dequeue();
            if((int) lastElement >= 0) {
                // it's positive, thus put the item back of the queue
                this.enqueue(lastElement);
            }
        }
    }
// Returns an iterator that iterates over the items in this queue in FIFO order.
    public Iterator<Item> iterator() {
        return new ListIterator();
    }
    // an iterator, doesn't implement remove() since it's optional
    private class ListIterator implements Iterator<Item> {
        private Node current = first;
        public boolean hasNext() { return current != null; }
        public void remove() { throw new UnsupportedOperationException(); }
        public Item next() {
            if (!hasNext()) throw new NoSuchElementException();
            Item item = current.item;
```

Queue Class (cont)

```
>           current = current.next;
        return item;
    }
  }

}
```

ArrayList data structure Commands

```
ArrayList<Integer> list = new ArrayList<>();
// Add an element to the list
list.a dd(x);
// Copy list
list.c lone();
// Element at the index i
list.g et(i);
// Is element present in the list, returns T/F
list.c ont ain s(x);
// Returns the index of the item x
list.i nde xOf(x);
// Returns the size of list
list.s ize();
// Check if list is empty, return T/F
list.i sEm pty();
// Replace an element x at index i with element y
list.s et(i, y);
// ArrayList into array:
size of array is same as the ArrayList
        int[] arr = new int[li st.s iz e()];
       // Convert ArrayList into an array
        lis t.t oAr ray (arr);
// Convert ArrayList into String, basically ouputs the list
list.t oSt ring();
```

Advanced Tutorial Content

Call Center Exercise

```
import java.util.ArrayList;
public class CallCe nter{
     // Constants for the simulation
      private final int number OfE mpl oyees = 10;
      private final int number OfT ime Periods = 480;
      private final int maxNum ber OfC ust omers = 5;
      private final int maxSer vic eTime = 5;
     // Variables to store simulation results
      private int totalW aitTime = 0;
      private int totalN umb erO fSe rve dCu stomers = 0;
      private int finalQ ueu eLength = 0;
      public static void main (String[] args){
            // Create an instance of the CallCenter class
             Cal lCenter myCall Center = new CallCe nter();
            // Run the call center simulation
             myC all Cen ter.ru nSi mul ati on();
            // Print the simulation statistics
             myC all Cen ter.pr int Sta tis tics();
     }
      public void printS tat ist ics(){
            // Printing the simulation statis tics.
             Sys tem.ou t.p rin tln ("The call center simulation was run with " + number OfE mpl oyees + "
employ ees.");
             Sys tem.ou t.p rin tln ("Be tween 9am and 5pm " + (final Que ueL ength + totalN umb erO fSe -
rve dCu sto mers) + " customers have called ");
             Sys tem.ou t.p rin tln ("The average wait time was "+ (1.0*t ota lWa itTime / totalN umb erO -
fSe rve dCu sto mers) + " minute s.");
             Sys tem.ou t.p rin tln ("Number of unserved customers: " + finalQ ueu eLe ngth);
     }
     // runs the simulation as described in the assignment
      public void runSim ula tion(){
            // We begin with an empty queue
             Que ue< Cus tom er> myQ = new Queue< >();
```

Call Center Exercise (cont)

```
>     // List of Employees
    ArrayList<Employee> listEmployees = assignEmployee(numberOfEmployees);
    // We go through the day:
    for(int currentTime = 0; currentTime < numberOfTimePeriods; currentTime++) {
      // Each minute new customers "join the queue"
      randomArrivals(maxServiceTime, maxNumberOfCustomers, currentTime, myQ);
      // For each employee we assign a new customer if possible:
      for(Employee employee : listEmployees) {
        // There are customers in queue and employee is available:
        if(employee.getAvailableAt() <= currentTime && !myQ.isEmpty()) {
          Customer servedCustomer = myQ.dequeue();
          // Adding how long customer waited to the total & recording him as served one.
          totalWaitTime += (currentTime - servedCustomer.getArrivalTime());
          totalNumberOfServedCustomers++;
          int timeRequired = servedCustomer.getServiceTime();
          // Change employees available time
          employee.setAvailableAt(currentTime + timeRequired);
        }
      }
      // After 8 hours, number of people not served:
      finalQueueLength = myQ.size();
    }
  }
  // helper method to fill employee list
  public ArrayList<Employee> assignEmployee(int n) {
    ArrayList<Employee> list = new ArrayList<>();
    for(int i = 0; i < n; i++) {
      // Create Employee objects and add them to the list
      Employee emp = new Employee(i);
      list.add(emp);
    }
```

Call Center Exercise (cont)

```
>       return list;
  }
  // helper method for simulating callers arriving to the queue:
  public void randomArrivals(int maxServiceTime, int maxNumberOfCustomers, int timeOfArrival, Queue myQ) {
    int numberOfArrivals = (int) (Math.random()*maxNumberOfCustomers + 1);
    for(int i = 0; i < numberOfArrivals; i++){
      // each customer will have different service time
      int serviceTime = (int) (Math.random()*maxServiceTime + 1);
      Customer customer = new Customer(timeOfArrival, serviceTime);
      // add customer to the end of the queue
      myQ.enqueue(customer);
    }
  }
}
```

Employee Class For Call Center

```
public class Employee{

    private final int id;
    private int availa bleAt;

   // constructs an Employee with an id and initial available time at 0
    public Employee (int id){
        this.id = id;
        thi s.a vai lableAt = 0;
   }

   // returns the id of this employee
    public int getId(){
        return this.id;
   }

   // returns the time at which an employee is available again
    public int getAva ila ble At(){
        return this.a vai lab leAt;
   }
```

Employee Class For Call Center (cont)

```
>  // sets the time at which an employee is available again
  public void setAvailableAt(int time){
     this.availableAt = time;
  }

}
```

Customer Class For Call Center

```
public class Customer{
    private final int arriva lTime;
    private final int servic eTime;

   // constructs a Customer with arrival time a and duration d
    public Customer (int arriva lTime, int servic eTime){
        thi s.a rri valTime = arriva lTime;
        thi s.s erv iceTime = servic eTime;
   }

   // returns the arrival time of this customer
    public int getArr iva lTi me(){
        return this.a rri val Time;
   }

   // returns the service time of this customer
    public int getSer vic eTi me(){
        return this.s erv ice Time;
   }

}
```

Elevator Problem

```
// method to solve the number of elevators problem:
        public static int number OfE lev ato rs( Arr ayL ist <Pe rso n> person List) {
                Arr ayL ist <In teg er> elevators = new ArrayL ist <>();
               // Initial " bin " is empty:
                ele vat ors.ad d(0);
               int elevat orCount = 1;
```

Elevator Problem (cont)

```
>       for(Person person:personList) {
         // flag will tell whether the person fitted in the current elevator:
         boolean flag = false;
         int weight = person.getWeight();
         // Check if person fits in one of the existing bins (elevators):
         for (int i = 0; i < elevatorCount; i++) {
            int currentElevatorWeight = elevators.get(i);
            // Person fits in the elevator
            if((currentElevatorWeight + weight) <= 450) {
               elevators.set(i, weight + currentElevatorWeight);
               flag = true;
            }
         }
         // Other elevators didn't have space, we open new "bin":
         if(!flag) {
            elevators.add(weight);
            elevatorCount++;
         }
      }
      return elevatorCount;
   }
// helper method to sort the list descending or ascending by weight. char d for descending and a for ascending.
   public static void insertionSortWeight(ArrayList<Person> personList, char order){
      int n = personList.size();
      for (int i = 1; i < n; i++) {
         Person key = personList.get(i);
         int j = i - 1;
         // Compare weights for sorting
         if (order == 'a' || order == 'A') { // Ascending order
            while (j >= 0 && personList.get(j).getWeight() > key.getWeight()) {
               personList.set(j + 1, personList.get(j));
```

Elevator Problem (cont)

```
>              j--;
            }
        } else if (order == 'd' || order == 'D') { // Descending order
            while (j >= 0 && personList.get(j).getWeight() < key.getWeight()) {
                personList.set(j + 1, personList.get(j));
                j--;
            }
        }
        personList.set(j + 1, key);
    }
}
```

Game Of Stacks

```
public int runGameStrategy1(){
                // The first step of the game is to create
the needed random stack
                 Int ege rStack gameStack = Intege rSt ack.cr eat eRa ndo mSt ack();
                // From here we go to the procedures of the game itself
                // generate two player scores and ID's
                int aliceID = 1;
                int bobID = -1;
                int aliceScore = 0;
                int bobScore = 0;
                // Move tracker that checks whos turn it is
                int move = 0;
                // The game starts
                 while (!game Sta ck.i sE mpt y()){
                        // First we check whos turn it is since both
players only take 1 turn every time due to the strategy
                        if (move % 2 == 0) {
                                // Alice starts and ONLY keeps every 1
                                if (gameS tac k.pop() == aliceID){
```

Game Of Stacks (cont)

```
>                   aliceScore += 1;
                }
            } else {
                // Bob starts and ONLY keeps every -1
                if (gameStack.pop() == bobID){
                    bobScore += 1;
                }
            }
            move += 1;
        }
        if (aliceScore > bobScore) {
            return 1;
        } else if (aliceScore == bobScore) {
            return 0;
        } else {
            return -1;
        }
    }
    public int runGameStrategy2() {
        // The first step of the game is to create
the needed random stack
        IntegerStack gameStack = IntegerStack.createRandomStack();
        int aliceScore = 0;
        int bobScore = 0;
        // Move tracker that checks whos turn it is
        int currentPlayer = 0;
        // The game starts
        while (!gameStack.isEmpty()) {
            int top = gameStack.pop();
            if( currentPlayer%2 == 0 ) {
                if (top == 1) {
```

Game Of Stacks (cont)

```
>                 int consecOnes = 1;
                  if(gameStack.isEmpty()) break;
                  int next = gameStack.pop();
                  if (next == 1) {
                     consecOnes++;
                  } else consecOnes = 0;
                  aliceScore += consecOnes;
               }
            } else if (top == -1) {
               bobScore += 1;
            }
            currentPlayer++;
         }
         if (aliceScore > bobScore) {
            return 1;
         } else if (aliceScore == bobScore) {
            return 0;
         } else {
            return -1;
         }
      }
```

Average BMI method

```
// Method that calculates average BMI index per age group:
        public static void averag eBM I(A rra yLi st< Per son> person List) {
                int[] ageGroup = {20,30 ,40 ,50 ,60 ,70};
               // 20-29 group corres ponds to index 0, 30-39 to 1 and so on
                int[] groupCount = new int[ag eGr oup.le ngth];
                dou ble[] groupS umBMI = new double [ag eGr oup.le ngth];
                for(int i = 0; i < person Lis t.s ize(); i++) {
                        Person curren tPerson = person Lis t.g et(i);
                       // age/10 - 2, will give correct index( group number) of the object for its place
in the arrays.
```

Average BMI method (cont)

```
>           int personsGroup = currentPerson.getAge()/10 - 2;
        groupCount[personsGroup]++;
        groupSumBMI[personsGroup] += currentPerson.getBMI();
    }
```

By **bennetsph**
cheatography.com/bennetsph/

Published 21st October, 2023.
Last updated 21st October, 2023.