

# בחינה בתוכנה 1

סמסטר ב', מועד א' 2013/~~2014~~

## ליאור ניר דביר ולנה

### הוראות (נא לקרוא!)

- משך הבחינה **שלוש שעות**, חלקו את זמנכם ביעילות.
- אסור השימוש בחומר עזר כלשהו, כולל מחשבוניו או כל מכשיר אחר פרט לעט. בסוף הבחינה צורף לנוחותכם נספח ובו תיעוד מחלקות שימושיות.
- יש לענות על כל השאלות בגוף הבחינה במקום המיועד לכך. המקום המיועד מספיק לתשובות מלאות. יש לצרף את טופס המבחן למחברת הבחינה. מחברת ללא טופס עזר תיפסל. תשובות במחברת הבחינה לא תיבדקנה. במידת הצורך ניתן לכתוב בגב טופס הבחינה.
- יש למלא מספר סידורי (מס' מחברת) ומספר ת.ז על כל דף של טופס הבחינה.
- ניתן להניח לאורך השאלה שכל החבילות הדרושות יובאו, ואין צורך לכתוב שורות import.
- במקומות בהם תתבקשו לכתוב מתודה (שירות), ניתן לכתוב גם מתודות עזר.
- ניתן להוסיף הנחות לגבי אופן השימוש בשירותים המופיעים בבחינה, ובלבד שאין הן סותרות את תנאי השאלה. יש לתעד הנחות אלו כחווה (תנאי קדם, תנאי בתר) בתחביר המקובל, שיכתב בתחילת השרות.
- יש להתייחס בכבוד רב לצוות המשגיחים.

לשימוש הבודקים בלבד:

שאלה	א	ב	ג	ד	ה	סה"כ
1						
2						
3						
4						

## בהצלחה!

כל הזכויות שמורות ©

מבלי לפגוע באמור לעיל, אין להעתיק, לצלם, להקליט, לשדר, לאחסן במאגר מידע, בכל דרך שהיא, בין מכונית ובין אלקטרונית או בכל דרך אחרת כל חלק שהוא מטופס הבחינה.

## שאלה 1 (35 נקודות)

משחק העצמות הוא משחק בו מטילים מספר קוביות שוב ושוב, כאשר בכל תור מוציאים מהמשחק ("הורגים") קוביות שערכיהן מקיימים חוקיות מסויימת.

המשחק מתחיל עם מספר כלשהו של קוביות ( $numLiveCubes=6$ ), כאשר כל קובייה מחזירה ערך רנדומאלי שלם בתחום 1-6 (הסתברות שווה לכל ערך). לאחר כל תור של הטלת הקוביות, הקוביות אשר לא עברו קריטריון מסויים מוצאות מהמשחק. ערכי כל הקוביות האחרות נסכמים, והסכום מתווסף למספר הנקודות הכולל של השחקן/ית (במשחק שלנו רק שחקן אחד).

לדוגמא:

חוק מקובל הוא שקוביות עם הערכים 2 ו-5 נפסלות. לפי חוק זה אם תוצאות הטלת הקוביות בסיבוב הראשון הן 1 2 3 4 5, אז שלוש מהקוביות (2 5 5) יפסלו, והניקוד לסיבוב יהיה  $8=1+3+4$ . בסיבוב הבא יוטלו שלוש קוביות. אם יצא 1 2 5, קוביה אחת תמשיך, והציון הכולל יהיה  $9=8+1$ . אם בסיבוב הבא הקוביה תראה 2 אז המשחק יגמר בציון 9.

מספר סיבוב	מספר קוביות פעילות	תוצאות ההטלה	ניקוד הטלה נוכחית	ניקוד בסיום הסיבוב
1	6	5 5 4 3 2 1	1+3+4	8
2	3	5 2 1	1	9
3	1	2	0	9

חוק מסוג זה, המתבסס על הערך של קובייה בודדת יקרא להלן **חוק פשוט**. חוק שמתבסס על ערכי כל הקוביות כדי לקבוע אם להעביר או לפסול קובייה ספציפית יקרא **חוק מורכב**. בשאלה זו נעסוק בין היתר בייצוג חוקים פשוטים (סעיף א') ובייצוג חוקים מורכבים (סעיף ב').

במהלך השאלה נשתמש באיטרטורים, כולל פונקצית `remove()`. תזכורת:

```
void remove ()
```

Removes from the underlying collection the last element returned by the iterator. This method can be called only once per call to `next`.

**סעיף א (20 נק'):** לפניכם שלד של פונקציה שמפעילה משחק עצמות. המתודה `Bones.playGame` מממשת את המשחק ומדפיסה לפני כל סיבוב את מספר הסיבוב, מספר הקוביות החיות בראשיתו, ואת הציון הכולל עד סיבוב זה (בסיבוב האחרון הציון אינו משתנה ולכן לא מדווחים בסוף סיבוב זה).

```
public class BonesRunner {
    public static void main(String[] args) {
        // RuleTwoFive represents the simple rule of killing 2 and 5
        // you need to define it yourself
        SimpleRule rule = new RuleTwoFive();
        Bones.playGame(6, rule);
    }
}
```

השלימו את הקוד בעמוד הבא כדי שהמשחק יתבצע לפי החוק הפשוט שהודגם למעלה (הוצאה של ערכי 2 ו-5). שימו לב, עליכם לממש קוד גם מחוץ למחלקה `Bones` (מותר להשתמש במחלקות פנימיות אם אתם בוחרים בכך).

```

import java.util.*;

public class Bones {

    // HINT: you need to define SimpleRule yourself
    public static void playGame(int numLiveCubes, SimpleRule rule) {
        int sum = 0;
        int round = 0;
        do {
            round++;
            System.out.println("Round: " + round + "; Cubes left: "
                + numLiveCubes + "; Sum: " + sum);
            List<Integer> cubes = rollCubes(numLiveCubes);
            filterCubes(cubes.iterator(), rule);
            for (int i : cubes)
                sum += i;
            numLiveCubes = cubes.size();
        } while (numLiveCubes > 0);
    }

    // Uses the supplied iterator to remove items that do not pass the // given rule
    private static void filterCubes(Iterator<Integer> cui,
        SimpleRule rule) {

        while (cui.hasNext()) {
            Integer i = cui.next();
            if (!rule.pass(i))
                cui.remove();
        }
    }

    // Generates numLiveCubes random integers with values 1..6
    // HINT: static double Math.random() Returns a double value in the
    // range [0,1). You may use other random functions.
    private static List<Integer> rollCubes(int numLiveCubes) {

        ArrayList<Integer> cubes = new ArrayList<Integer>();
        for (int i = 0; i < numlivecubes; i++)
            cubes.add(1 + (int) (Math.random() * ((5) + 1)));
        return cubes;
    }
}

```

```
public class RuleTwoFive implements SimpleRule {  
    public boolean pass(int i) {  
        return ((i != 2) && (i != 5));  
    }  
}  
  
interface SimpleRule {  
    boolean pass(int i);  
}
```

**סעיף ב (15 נק'): עתה נממש חוק מורכב. בכל סיבוב, הקובייה עם הערך הכי גבוה מכל הקוביות בהטלה הנוכחית תפסל. למשל עבור הקוביות 1 2 3 4 4 4, הציון יהיה  $6=1+2+3$  ובסיבוב הבא יוגרלו שלוש קוביות. אם בסיבוב הבא יצא 1 1 1, אז המשחק יגמר בציון 6.**

```
public class BonesRunner2 {
    public static void main(String[] args) {
        // Represents the complex rule of killing the max value
        // the use of SimpleRule as the type here is not a bug
        SimpleRule rule = new RuleKillMaxVal();
        Bones.playGame(6, rule);
    }
}
```

עליכם לשנות את המחלקה Bones וליצור מחלקות נוספות לפי הצורך. (פונקציות שלא משתנות בהכרח מהגירסא בסעיף א' לא מופיעות למטה). שימו לב, חוק מורכב מצריך איטרטור נוסף כדי לספק סטטיסטיקות על הערכים בסיבוב המשחק הנוכחי. שימו לב, המחלקה Bones שומרת על תאימות אחורה לחוקים פשוטים.

```
public class Bones {
    public static void playGame(int numLiveCubes, SimpleRule rule) {
        int sum = 0;
        int round = 0;
        do {
            round++;
            System.out.println("Round: " + round + "; Cubes left: "
                + numLiveCubes + "; Sum: " + sum);
            ArrayList<Integer> cubes = rollCubes(numLiveCubes);
            if ( !(rule instanceof ComplexRule) )
                filterCubes(cubes.iterator(), rule);
            else
                filterCubesComplex(
                    cubes.iterator(),
                    cubes.iterator(),
                    (ComplexRule) rule );
            for (int i : cubes)
                sum += i;
            numLiveCubes = cubes.size();
        } while (numLiveCubes > 0);
    }
}
```

```
private static void filterCubesComplex(
    Iterator<Integer> cuiforstats,
    Iterator<Integer> cui,
```

```
ComplexRule rule) {
```

```
    rule.init(cuiforstats);
    filterCubes(cui, rule);
```

```
}
```

```
}
```

```
public class RuleKillMaxVal implements ComplexRule {
```

```
    int badi;
```

```
    public void init(Iterator<Integer> I) {
```

```
        badi = 0;
```

```
        while (I.hasNext()) {
```

```
            int i = I.next();
```

```
            if (i > badi)
```

```
                badi = i;
```

```
        }
```

```
    };
```

```
    public boolean pass(int i) {
```

```
        return (i != badi);
```

```
    }
```

```
}
```

```
interface ComplexRule extends SimpleRule {
```

```
    void init(Iterator<Integer> I);
```

```
}
```

**שאלה 2 (20 נקודות)**

הסעיפים בשאלה זו מתייחסים לשלוש המחלקות הבאות:

```

public class A {
    public void foo(A a1, A a2) {
        System.out.println("A" + ((A)a1).foo());
    }

    public int foo() {
        return 1;
    }
}

public class B extends A {
    public void foo(A a1, A a2) {
        System.out.println("B" + ((B)a1).foo());
    }
    public int foo() {
        return 2;
    }
}

public class C extends B {

    *****

    public static void main(String[] args) {
        A a = new A();
        A b = new B();
        C c = new C();
        c.foo(b, a);
    }
}

```

בכל אחד מהסעיפים הבאים מוחלפת שורת הכוכביות בקטע קוד. הנכם מתבקשים לציין מהו הפלט של הרצת פונקציית ה main של המחלקה C בכל אחד מהמקרים. אם לדעתכם אין פלט לתוכנית בשל בעיית קומפילציה או שגיאה בזמן ריצה (זריקת חריג) – הסבירו מה השגיאה. פתרון ללא הסבר לא יזכה בנקודות.

**סעיף א (4 נק'):**

```

public int foo(){
    return 3;
}

```

**תשובה:**

B2

סעיף ב (4 נק'):

```
public void foo(A a1, A a2){
    super.foo(a2, a2);
}
```

תשובה:

נזרקה הודעת שגיאה `class cast Exception`.

סעיף ג (4 נק'):

```
public void foo(A a1, A a2){
    a2.foo(a1, a1);
}
```

תשובה:

A2

סעיף ד (4 נק'):

```
public int foo(A a1, A a2) {
    System.out.println("C");
    return 5;
}
```

תשובה:

שגיאת קומפילציה בגלל שערך ההחזרה של `foo` הראשונה שונה מזה של המתודה במחלקה B

סעיף ה (4 נק'):

```
public void foo(A a1, A a2){
    foo(a1, a1);
    System.out.println("C");
}
```

תשובה:

לולאה אינסופית של קריאות רקורסיביות שבסופה תיזרק הודעת שגיאה `StackOverflowError`. לא תתבצע אף הדפסה.



**שאלה 3 (20 נקודות)**

הסעיפים בשאלה זו מתייחסים לתאים ברשימה מקושרת המוגדרים במחלקות הבאות:

```
public class Link {
    public static Link LAST = new Link(-1);

    public Link next;
    public int myId;

    public Link() {
        this(10);
    }

    public Link(Link next) {
        this(20);
        this.next = next;
    }

    protected Link(boolean b) {
        this(99);
    }

    private Link(int myId) {
        this.myId = myId;
    }

    @Override
    public String toString() {
        return "Link #" + myId;
    }
}

public class DataLink<T> extends Link {
    private T value;

    public DataLink(T value) {
        this(value, new Link(false));
    }

    public DataLink(T value, Link parent) {
        super(parent);
        this.value = value;
    }

    @Override
    public String toString() {
        return super.toString() + " (" + value.toString() + ")";
    }
}

public class Main {
    public static void main(String[] args) {
        *****
    }
}
```

בכל אחד מהסעיפים הבאים מוחלפת שורת הכוכביות בקטע קוד. הנכם מתבקשים לציין מהו הפלט של הרצת פונקציית ה main של המחלקה Main בכל אחד מהמקרים.  
 אם לדעתכם אין פלט לתוכנית בשל בעיית קומפילציה או שגיאה בזמן ריצה (זריקת חריג) – הסבירו מה השגיאה. פתרון ללא הסבר לא יזכה בנקודות.

**סעיף א (4 נק'):**

```
System.out.println(Link.LAST.next);
```

תשובה:

```
null
```

**סעיף ב (4 נק'):**

```
System.out.println(new Link());
```

תשובה:

```
Link #10
```

**סעיף ג (4 נק'):**

```
System.out.println(new DataLink<>(1));
```

תשובה:

```
Link #20 (1)
```

**סעיף ד (4 נק'):**

```
System.out.println(new DataLink<>(new Link()).next);
```

תשובה:

```
Link #99
```

**סעיף ה (4 נק'):**

```
System.out.println(new Link(1));
```

תשובה:

שגיאת קומפילציה, הבנאי בניראות private

## שאלה 4 (25 נקודות)

ממשו מתודה המקבלת שמות של שני קבצי טקסט ומחזירה את רשימת המילים מהקובץ הראשון, שנמצאות גם בקובץ השני. על המילים ברשימה המוחזרת להיות ממוינות על פי מספר המופעים שלהן בקובץ הראשון (במקום הראשון ברשימה המוחזרת תופיע המילה הנפוצה ביותר בקובץ הראשון, שנמצאת גם ברשימת המילים שבקובץ השני). הנה חתימת המתודה:

```
public static List<String> filterAndSortByFrequency (String inputFileName,
String wordsFileName)
```

כלומר, נחשב את מספר המופעים של כל מילה ב-inputFileName שנמצאת ברשימת המילים שבקובץ wordsFileName. ניתן להיעזר בתקציר הפקודות המופיע בסוף הבחינה. הערות:

- הקובץ wordsFileName יכיל רשימת מילים ללא חזרות, כל מילה בשורה נפרדת.
- ניתן להניח שהמילים בשני הקבצים מורכבות מאותיות לועזיות בלבד, ומופרדות ע"י רווחים או ע"י ירידות שורה בלבד.
- ניתן להניח ששני הקבצים נמצאים בתיקייה בה תרוץ התוכנית (אין צורך לשנות נתיב).
- במידה ויש כמה מילים עם אותו מספר מופעים, אין חשיבות לסדר ההחזרה שלהן.
- ניקוד מלא יינתן למימוש אלגנטי העושה שימוש באוספים (כגון Map, Set ...) וכולל מעבר יחיד על כל אחד מהקבצים (פותרים וקוראים כל קובץ רק פעם אחת).
- במקרה של בעיה כלשהי במהלך ריצת המתודה (חריגה), המתודה תחזיר null.

```

public static List<String> filterAndSortByFrequency (String inputFileName,
                                                    String wordsFileName) {

    Scanner wordsScanner = null;
    Scanner inputScanner = null;

    try {

        // Filling a HashSet with words from the words file
        HashSet<String> wordsSet = new HashSet<String>();
        wordsScanner = new Scanner (new File(wordsFileName));
        while (wordsScanner.hasNext()) {
            wordsSet.add(wordsScanner.next());
        }

        // Counting word occurrences in the input file using a HashMap
        HashMap<String,Integer> wordsFreq = new HashMap<String,Integer>();
        inputScanner = new Scanner (new File(inputFileName));
        while (inputScanner.hasNext()) {
            String curWord = inputScanner.next();
            if (wordsSet.contains(curWord)) { // Filtering words
                if (!wordsFreq.containsKey(curWord))
                    wordsFreq.put (curWord, 1);
                else
                    wordsFreq.put (curWord, wordsFreq.get (curWord)+1);
            }
        }

        // Sorting the map's entries by the value
        List<Map.Entry<String,Integer>> entries=
            new ArrayList<Map.Entry<String,Integer>>( wordsFreq.entrySet());
        Collections.sort(entries, new Comparator<Map.Entry<String, Integer>>() {

            @Override
            public int compare(Entry<String, Integer> o1, Entry<String, Integer> o2) {
                // descending order
                return ((Integer) o2.getValue()) - ((Integer) o1.getValue());
            }
        });

        // Builds the result list composed of only keys
        List<String> result = new ArrayList<String>();
        for (int i=0; i<entries.size(); i++)
            result.add((String) entries.get(i).getKey());

        return result;
    } catch (Exception e) {
        return null;
    }
    finally {
        if (wordsScanner!=null)
            wordsScanner.close();
        if (inputScanner!=null)
            inputScanner.close();
    }
}

```

## נספח

### File

#### Constructor Summary

[File](#)([String](#) pathname)

Creates a new `File` instance by converting the given pathname string into an abstract pathname.

### FileReader

#### Constructor Summary

[FileReader](#)([File](#) file)

Creates a new `FileReader`, given the `File` to read from.

[FileReader](#)([String](#) fileName)

Creates a new `FileReader`, given the name of the file to read from.

#### Method Summary

##### Methods inherited from class [java.io.InputStreamReader](#)

[close](#), [getEncoding](#), [read](#), [read](#), [ready](#)

##### Methods inherited from class [java.io.Reader](#)

[mark](#), [markSupported](#), [read](#), [read](#), [reset](#), [skip](#)

### BufferedReader

#### Constructor Summary

[BufferedReader](#)([Reader](#) in)

Create a buffering character-input stream that uses a default-sized input buffer.

#### Method Summary

void	<a href="#">close</a> ()	Close the stream.
int	<a href="#">read</a> ()	Read a single character.
int	<a href="#">read</a> (char[] cbuf, int off, int len)	Read characters into a portion of an array.
<a href="#">String</a>	<a href="#">readLine</a> ()	Read a line of text.
boolean	<a href="#">ready</a> ()	

	Tell whether this stream is ready to be read.
void	<a href="#">reset</a> () Reset the stream to the most recent mark.
long	<a href="#">skip</a> (long n) Skip characters.

## Scanner

### Constructor Summary

<a href="#">Scanner</a> ( <a href="#">File</a> source)	Constructs a new <code>Scanner</code> that produces values scanned from the specified file.
<a href="#">Scanner</a> ( <a href="#">InputStream</a> source)	Constructs a new <code>Scanner</code> that produces values scanned from the specified input stream.
<a href="#">Scanner</a> ( <a href="#">Readable</a> source)	Constructs a new <code>Scanner</code> that produces values scanned from the specified source.
<a href="#">Scanner</a> ( <a href="#">String</a> source)	Constructs a new <code>Scanner</code> that produces values scanned from the specified string.

### Method Summary

void	<a href="#">close</a> () Closes this scanner.
<a href="#">Pattern</a>	<a href="#">delimiter</a> () Returns the <code>Pattern</code> this <code>Scanner</code> is currently using to match delimiters.
boolean	<a href="#">hasNext</a> () Returns true if this scanner has another token in its input.
boolean	<a href="#">hasNext</a> ( <a href="#">String</a> pattern) Returns true if the next token matches the pattern constructed from the specified string.
boolean	<a href="#">hasNextLine</a> () Returns true if there is another line in the input of this scanner.
<a href="#">IOException</a>	<a href="#">ioException</a> () Returns the <code>IOException</code> last thrown by this <code>Scanner</code> 's underlying <code>Readable</code> .
<a href="#">String</a>	<a href="#">next</a> () Finds and returns the next complete token from this scanner.
<a href="#">String</a>	<a href="#">nextLine</a> () Advances this scanner past the current line and returns the input that was skipped.
<a href="#">Scanner</a>	<a href="#">useDelimiter</a> ( <a href="#">String</a> pattern) Sets this scanner's delimiting pattern to a pattern constructed from the specified <code>String</code> .

# Collections

Interface	Common implementations
List<E>	ArrayList<E>, LinkedList<E>
Set<E>	HashSet<E>, SortedSet<E>
Map<K, V>	HashMap<K, V>, TreeMap<K, V>

## Interface List<E>

Method Summary	
boolean	<a href="#">add</a> (E e)
void	<a href="#">add</a> (int index, E element)
boolean	<a href="#">addAll</a> (Collection<? extends E> c)
boolean	<a href="#">addAll</a> (int index, Collection<? extends E> c)
void	<a href="#">clear</a> ()
boolean	<a href="#">contains</a> (Object o)
boolean	<a href="#">containsAll</a> (Collection<?> c)
boolean	<a href="#">equals</a> (Object o)
E	<a href="#">get</a> (int index)
int	<a href="#">hashCode</a> ()
int	<a href="#">indexOf</a> (Object o)
boolean	<a href="#">isEmpty</a> ()
Iterator<E>	<a href="#">iterator</a> ()
int	<a href="#">lastIndexOf</a> (Object o)
ListIterator<E>	<a href="#">listIterator</a> ()
ListIterator<E>	<a href="#">listIterator</a> (int index)
E	<a href="#">remove</a> (int index)
boolean	<a href="#">remove</a> (Object o)
boolean	<a href="#">removeAll</a> (Collection<?> c)
boolean	<a href="#">retainAll</a> (Collection<?> c)
E	<a href="#">set</a> (int index, E element)
int	<a href="#">size</a> ()
List<E>	<a href="#">subList</a> (int fromIndex, int toIndex)
Object[]	<a href="#">toArray</a> () (from first to last element).
<T> T[]	<a href="#">toArray</a> (T[] a)

## Interface Set<E>

Method Summary	
boolean	<a href="#">add</a> (E e)
boolean	<a href="#">addAll</a> (Collection<? extends E> c)
void	<a href="#">clear</a> ()
boolean	<a href="#">contains</a> (Object o)

boolean	<a href="#">containsAll</a> ( <a href="#">Collection</a> <?> c)
boolean	<a href="#">equals</a> ( <a href="#">Object</a> o)
int	<a href="#">hashCode</a> ()
boolean	<a href="#">isEmpty</a> ()
<a href="#">Iterator</a> <E>	<a href="#">iterator</a> ()
boolean	<a href="#">remove</a> ( <a href="#">Object</a> o)
boolean	<a href="#">removeAll</a> ( <a href="#">Collection</a> <?> c)
boolean	<a href="#">retainAll</a> ( <a href="#">Collection</a> <?> c)
int	<a href="#">size</a> ()
<a href="#">Object</a> []	<a href="#">toArray</a> ()
<T> T[]	<a href="#">toArray</a> (T[] a)

## Interface Map<K,V>

Method Summary	
void	<a href="#">clear</a> ()
boolean	<a href="#">containsKey</a> ( <a href="#">Object</a> key)
boolean	<a href="#">containsValue</a> ( <a href="#">Object</a> value)
<a href="#">Set</a> < <a href="#">Map.Entry</a> <K,V>>	<a href="#">entrySet</a> ()
boolean	<a href="#">equals</a> ( <a href="#">Object</a> o)
V	<a href="#">get</a> ( <a href="#">Object</a> key)
int	<a href="#">hashCode</a> ()
boolean	<a href="#">isEmpty</a> ()
<a href="#">Set</a> <K>	<a href="#">keySet</a> ()
V	<a href="#">put</a> (K key, V value)
void	<a href="#">putAll</a> ( <a href="#">Map</a> <? extends K,? extends V> m)
V	<a href="#">remove</a> ( <a href="#">Object</a> key)
int	<a href="#">size</a> ()
<a href="#">Collection</a> <V>	<a href="#">values</a> () Returns a <a href="#">Collection</a> view of the values contained in this map.

Nested Class Summary	
static interface	<a href="#">Map.Entry</a> <K,V> A map entry (key-value pair).

## Class Collections

Method Summary	
static <T> boolean	<a href="#">addAll</a> ( <a href="#">Collection</a> <? super T> c, T... elements)
static <T> void	<a href="#">copy</a> ( <a href="#">List</a> <? super T> dest, <a href="#">List</a> <? extends T> src)
static <T> <a href="#">Enumeration</a> <T>	<a href="#">enumeration</a> ( <a href="#">Collection</a> <T> c)
static	<a href="#">fill</a> ( <a href="#">List</a> <? super T> list, T obj)



<T> void	
static int	<a href="#">frequency</a> ( <a href="#">Collection</a> <?> c, <a href="#">Object</a> o)
static int	<a href="#">indexOfSubList</a> ( <a href="#">List</a> <?> source, <a href="#">List</a> <?> target)
static int	<a href="#">lastIndexOfSubList</a> ( <a href="#">List</a> <?> source, <a href="#">List</a> <?> target)
static <T> <a href="#">ArrayList</a> <T>	<a href="#">list</a> ( <a href="#">Enumeration</a> <T> e)
static void	<a href="#">reverse</a> ( <a href="#">List</a> <?> list)
static <T extends <a href="#">Comparable</a> <? super T>> void	<a href="#">sort</a> ( <a href="#">List</a> <T> list) Sorts the specified list into ascending order, according to the <i>natural ordering</i> of its elements.
static <T> void	<a href="#">sort</a> ( <a href="#">List</a> <T> list, <a href="#">Comparator</a> <? super T> c) Sorts the specified list according to the order induced by the specified comparator.