

פתרון המבחן בתוכנה 1

סמסטר א', מועד א', תשע"ה

10.02.2015

שחר מעוז, יעל אמסטרדמר, דביר נתנאלי, לנה דנקין

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 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 כחוזה (תנאי קדם, תנאי בתר) בתחביר המקובל, שייכתב בתחילת השרות.

לשימוש הבודקים בלבד:

שאלה	משקל	א	ב	ג	ד	ה	סה"כ
1	40						
2	20						
3	20						
4	20						

בהצלחה!

© כל הזכויות שמורות

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

שאלה 1 (40 נקודות)

בשאלה זו נכתוב מחלקות גנריות המייצגות קבוצת איברים (ללא סדר וללא כפילויות) עם היסטוריה. כלומר, עבור הקבוצה נשמרות כל הגרסאות הקודמות שלה. המחלקות יממשו את הממשק הגנרי:

```
public interface ISetWithHistory<E> {
    public boolean addElement(E element);
    public boolean removeElement(E element);
    public ISetWithHistory<E> getPreviousSet();
}
```

- הממשק גנרי, כלומר, איבריו יכולים להיות מכל טיפוס.
- המתודות addElement ו-removeElement מחזירות true אם הוספת/הורדת איבר שינתה בפועל את איברי הקבוצה. (למשל, הורדת איבר שאינו נמצא בקבוצה אינה משנה את הקבוצה.)
- המתודה getPreviousSet() מחזירה הפניה לקבוצה כפי שהייתה לפני השינוי האחרון. אם לא הייתה גרסה קודמת, יוחזר הערך null.
- גרסה חדשה נוצרת רק אם בוצע שינוי בפועל של איברי הקבוצה ע"י הוספת או הורדת איבר.

הקוד הבא מובא כדוגמה לשימוש במחלקות המממשות את הממשק:

```
ISetWithHistory<String> set = ...;
System.out.println(set.getPreviousSet()); // prints null

set.addElement("A"); // returns true
set.addElement("B"); // returns true
set.addElement("B"); // returns false
System.out.println(set); // prints [A, B]
System.out.println(set.getPreviousSet()); // prints [A]

set.removeElement("C"); // returns false
set.removeElement("A"); // returns true
System.out.println(set); // prints [B]
System.out.println(set.getPreviousSet()); // prints [A, B]
System.out.println(set.getPreviousSet().getPreviousSet()); // prints [A]
```

א. (4 נק') מימוש מחלקת אב אבסטרקטית

נשים לב כי כבר קיימים בספרייה הסטנדרטית אוספים המאפשרים לשמור איברים ללא חזרות, אולם הם מממשים ממשק שונה (`java.util.Set<E>`). השלימו את המחלקה `AbstractSetWithHistory`, המממשת את הממשק `ISetWithHistory<E>` ומשתמשת בהכלה ובהאצלה של אוספים קיימים כדי לממש פעולות של הוספה, הסרה והמרה למחרוזת של איברי `SetWithHistory`. ניתן להיעזר בנספח לתיאור המתודות של `Set`.

שימו לב, בשלב זה המחלקה עדיין אינה מממשת את מנגנון שמירת ההיסטוריה.

```
public abstract class AbstractSetWithHistory<E> implements ISetWithHistory<E> {
```

```
// fields
protected final Set<E> elements = new HashSet<E>();
```

```
public boolean addElement(E element) {
    return elements.add(element);
```

```
}
```

```
public boolean removeElement(E element) {
    return elements.remove(element);
```

```
}
```

```
public abstract ISetWithHistory<E> getPreviousSet();
```

```
public String toString() {
    return elements.toString();
```

```
}
```

```
}
```

ב. (10 נק') מימוש שמירת היסטוריה באופן נאיבי

בעמוד הבא, השלימו את מימוש המחלקה `NaiveSetWithHistory<E>`, אשר יורשת מ-`AbstractSetWithHistory` ומוסיפה לה שמירת היסטוריית גרסאות "נאיבית":

עם כל הורדה והוספה של איבר המשנות את הקבוצה, המופע של הקבוצה ייצור עותק של כל איברי הקבוצה לפני השינוי, ויוסיף את העותק הזה לשדה המייצג את היסטוריית הגרסאות. כדי לייצר גרסא קודמת של הקבוצה, ניצור קבוצה חדשה עם האיברים מהגרסא הקודמת (גרסא אחת לפני הנוכחית) ועם כל ההיסטוריה ללא הגרסא האחרונה. ניתן להיעזר במתודה `deepCloneCurrentSet`, אשר מייצרת עותק זהה של קבוצה כולל ההיסטוריה, וניתן להניח שכבר מומשה עבורכם.

אין צורך לממש את `removeElement` (מימוש זה דומה ל-`addElement`).

```

public class NaiveSetWithHistory<E> extends AbstractSetWithHistory<E> {

    // fields
    // can also be implemented as a List
    private Deque<Set<E>> setHistory = new LinkedList<>();

    public boolean addElement(E element) {
        if (!this.elements.contains(element)) {
            // only if the element set changes --
            // add a clone of the current element set to the history record
            setHistory.add(new HashSet<E>(this.elements));
            return super.addElement(element);
        }
        return false;
    }

    public ISetWithHistory<E> getPreviousSet() {
        if (setHistory.isEmpty()) {
            return null;
        }
        // The previous set is a clone of the current set...
        NaiveSetWithHistory<E> prevSet = deepCloneCurrentSet();

        // ... minus the last change
        Set<E> lastSet = prevSet.setHistory.removeLast();
        prevSet.elements.clear();
        prevSet.elements.addAll(lastSet);

        return prevSet;
    }

    /** Creates and returns a "safe clone" of this */
    private NaiveSetWithHistory<E> deepCloneCurrentSet() {
        /* No need to implement */
    }

    public boolean removeElement(E element) { /* No need to implement */
    }
}

```

ג. (12 נק') מימוש יעיל יותר של שמירת ההיסטוריה

כדי להימנע משמירת עותק מלא של הקבוצה בכל גרסא, נרצה לשמור רק מהו השינוי שביצענו בכל גרסא. כדי לאחסן את מאפייני השינויים שבוצעו (סוג השינוי והערך עליו בוצע השינוי), תכלול המחלקה `EfficientSetWithHistory` את הרכיבים הבאים:

- מחלקה פנימית מסוג אנומרציה (enum) המייצגת את סוג השינוי:

```
private enum ChangeType {
    ADD, REMOVE //we can add here other change types
};
```

- מחלקה פנימית שתשמש לשמירת זוגות של סוג השינוי והערך ששונה:

```
private class ChangeValuePair {
    private ChangeType change;
    private E value;

    public ChangeValuePair(ChangeType change, E value) {
        this.change = change;
        this.value = value;
    }
}
```

במחלקה `EfficientSetWithHistory` נשמור את היסטוריית כל השינויים שביצענו לפי הסדר (כמו בסעיפים הקודמים, רק את אלה שגרמו לשינוי בקבוצת האיברים בפועל!). כדי לייצר גרסא קודמת של הקבוצה, ניצור תחילה עותק זהה שלה (כולל ההיסטוריה) ואז נשתמש במתודת העזר `reverseLastChange` כדי לבטל את הפעולה האחרונה שקרתה, ולמחוק אותה מההיסטוריה.

בהמשך ובעמוד הבא **שבעה** מלבנים ריקים להשלמת קוד המחלקה. ייתכן שלא תזדקקו לכולם. גם כאן, אין צורך לממש את `removeElement`.

```
public class EfficientSetWithHistory<E> extends
    AbstractSetWithHistory<E> implements ISetWithHistory<E> {
```

```
// can also be implemented as a List
private Deque<ChangeValuePair> changeHistory =
    new LinkedList<ChangeValuePair>();
```

```
public boolean addElement(E element) {
    if (!this.elements.contains(element)) {
        // only if the element set changes --
        // add a change to the history record
        changeHistory.add(new ChangeValuePair(ChangeType.ADD,
            element));
        return super.addElement(element);
    }
    return false;
}
```

```
}
```

```
public boolean removeElement(E element) { /* No need to implement */}
```

```

public ISetWithHistory<E> getPreviousSet() {
    if (changeHistory.isEmpty()) {
        return null;
    }
    // The previous set is a clone of the current set...
    EfficientSetWithHistory<E> prevSet = deepCloneCurrentSet();

    // ... minus the last change
    prevSet.reverseLastChange();

    return prevSet;
}

/** creates and returns a "safe clone" of this */
private EfficientSetWithHistory<E> deepCloneCurrentSet() {
    /* No need to implement */
}

/** Reverses the last change in the history of this set
    @pre the history is not empty */
private void reverseLastChange() {

ChangeValuePair lastChange = changeHistory.removeLast();
switch (lastChange.change) {
case ADD:
    super.removeElement(lastChange.value);
    break;

case REMOVE:
    super.addElement(lastChange.value);
    break;
}
}

private enum ChangeType {
    ADD, REMOVE //we can add here other change types
};

private class ChangeValuePair {
    private ChangeType change;
    private E value;
    public ChangeValuePair(ChangeType change, E value) {
        this.change = change;
        this.value = value;
    }
}

```

}

ד. (14 נקודות) על המחלקות המממשות `ISetWithHistory<E>` לספק איטרטור לעצמים מסוגן. האיטרטור יתקדם אחורה בזמן על הגרסאות ההיסטוריות של הקבוצה, מהנוכחית ועד לקדומה ביותר.

נוסיף לממשק `ISetWithHistory<T>` את השירות

```
public HistoryIterator<E> getHistoryIterator();
```

במחלקה `AbstractSetWithHistory<E>` השירות ימומש כך:

```
public HistoryIterator<E> getHistoryIterator() {
    return new HistoryIterator<E>(this);
}
```

כתבו מימוש לאיטרטור `HistoryIterator<E>`, אשר מממש את הממשק `Iterator` מעל עצמים מסוג `ISetWithHistory`

אין צורך לממש את מתודת `.remove()`

```
public class HistoryIterator<E> implements Iterator<ISetWithHistory<E>> {
```

```
    private ISetWithHistory<E> currentSet;
```

```
    HistoryIterator(ISetWithHistory<E> currentSet) {
        this.currentSet = currentSet;
    }
```

```
    public boolean hasNext() {
        return (currentSet != null);
    }
```

```
    public ISetWithHistory<E> next() {
        ISetWithHistory<E> temp = currentSet;
        currentSet = currentSet.getPreviousSet();
        return temp;
    }
```

}

שאלה 2 (20 נקודות)

סעיף א' (10 נק):

עבור מחרוזת `str` כלשהי, רצף של `k` תווים מתוכה מכונה "תת מחרוזת באורך `k`".

ממשו את הפונקציה הסטטית `getHistogram` אשר מקבלת כפרמטר מחרוזת `str` ומספר שלם `k`, ומחזירה אובייקט מטיפוס `Map<String, Integer>`, כאשר מפתחות המפה הם כל תתי המחרוזות (החופפות) של `str` באורך `k`, וערכי המפה הם מספר המופעים של אותה תת מחרוזת ב `str` (מיפוי זה נקרא היסטוגרמה). על ההיסטוגרמה להכיל אך ורק תתי מחרוזות באורך `k` אשר מופיעות ב `str`.

הפונקציה תזרוק חריג מטיפוס `InvalidValueException` כאשר ערכו של `k` גדול מאורך המחרוזת או קטן מ 1.

הניחו שהמחלקה `InvalidKValueException` מוגדרת באופן הבא:

```
public class InvalidKValueException extends Exception{
    public InvalidKValueException(){
        super("Invalid K value");
    }
}
```

דוגמא - עבור המחרוזת "ababaca":

תוכן הטבלה (key->value)	ערכו של k
"a" -> 4 , "b" -> 2 , "c" -> 1	1
"ab" -> 2 , "ba" -> 2 , "ac" -> 1 , "ca" -> 1	2
"aba" -> 2 , "bab" -> 1 , "bac" -> 1 , "aca" -> 1	3

```
/*
 * @pre str != null
 */
public static Map<String, Integer> getHistogram(String str, int k) throws
InvalidKValueException {
```

```
    if (k > str.length() || k <= 0){
        throw new InvalidKValueException ();
    }
    Map<String, Integer> res = new HashMap<String, Integer>();
    for (int i = 0; i <= str.length() - k; i++){
        String key = str.substring(i, i+k);
        if (res.containsKey(key)){
            res.put(key, res.get(key)+1);
        }
        else{
            res.put(key, 1);
        }
    }
    return res;
}
```

```
}
```


סעיף ב' (10 נק'): _____

הגדרה: שתי מחרוזות str1 ו-str2 הן **k-שוות** אם כל תת מחרוזת באורך k של str1 מופיעה גם ב-str2 ולהפך. מספר המופעים של תתי המחרוזות לא צריך להיות זהה בין str1 ל str2.

נגדיר גם שכל זוג מחרוזות הן 0-שוות.

ממשו את הפונקציה הסטטית getMaxK אשר בהינתן שתי מחרוזות str1 ו str2 תחזיר k **מקסימלי** (נקרא לו maxK) עבורו שתי המחרוזות הן k-שוות. אין לשנות את חתימת המתודה. ניתן להניח שהמתודה תופיע באותה מחלקה בה מימשתם את המתודה בסעיף א'.

דוגמא:

הסבר	maxK	str2	str1
תתי המחרוזות באורך 1 הן {"a", "b"}	1	"ab"	" <u>aa</u> b"
	0	" <u>d</u> abc"	"abc"
תתי המחרוזות באורך 3 הן {"bab", "aba", "bac", "aca"}	3	"babaca"	" <u>abab</u> aca"
תתי המחרוזות באורך 2 הן {"aa"}	2	"aa"	" <u>aaaaaa</u> "
	3	"abc"	"abc"

בכל שורה, מודגשת (אם קיימת) תת מחרוזת שאורכה maxK+1 ומופיעה רק באחת המחרוזות.

```

/*
 * @pre str1 != null && str2 != null
 */
public static int getMaxK(String str1, String str2) {

```

```

    int maxK = 0;
    int k=1;
    while (true){
        try{
            Set<String> set1 = getHistogram(str1, k).keySet();
            Set<String> set2 = getHistogram(str2, k).keySet();
            Set<String> unionSet = new HashSet<String>(set1);
            unionSet.addAll(set2);
            if (set1.size() == set2.size() && unionSet.size() == set1.size()){
                maxK=k;
                k++;
            }
            else{
                break;
            }
        }
        catch(InvalidKValueException e){
            break;
        }
    }
    return maxK;

```

}

שאלה 3 - GUI (20 נקודות)

בשאלה זו נשלים קוד של ממשק GUI פשוט ב-SWT. התכנית PressItGame מקבלת כקלט מהמשתמש דרך שורת הפקודה מספר שלם חיובי קטן, אשר ייקבע את מספר הכפתורים בחלון המשחק שייפתח. מהלך המשחק הוא כדלקמן:

i. למשתמש מוצגת הודעה בראש החלון עם מספר הכפתור הבא שעליו יש ללחוץ.

ii. אם המשתמש לוחץ על הכפתור הנכון, המשחק ממשיך ומוצגת ההודעה ללחוץ על הכפתור הבא.

iii. אם הוא לוחץ על כפתור שגוי, מוצגת למשתמש הודעה "GAME OVER" בראש החלון, ומאותה נקודה של סיום המשחק והלאה לחיצות נוספות על כפתורים אינן משנות דבר.

לדוגמא: כאשר הארגומנט של התכנית הוא 7, אחד ממהלכי המשחק האפשריים הוא (מימין לשמאל):



בקוד התכנית למטה (הממשיך בעמוד הבא) ישנם שני מקטעי קוד חסרים A ו-B. בנוסף, קטע הקוד הסטנדרטי של לולאת האירועים מושמט, וניתן להניח שהוא כתוב במקום המתאים במתודת ה-main.

אנו נכתוב השלמה למקטע הקוד הראשון ושלוש השלמות חלופיות (אלטרנטיביות) למקטע הקוד השני שיגדירו דרכים שונות לבחירת הכפתור הבא ללחיצה.


שימו לב: בכל הסעיפים א'-ד' מומלץ להשתמש במתודות העזר ובשדות הנתונים במחלקה במידת האפשר.

```
public class PressItGame {
    // fields
    private static int numButtons;
    private static Label message;
    private static boolean gameOver = false;
    private static int next; // The number of the next button to press
```

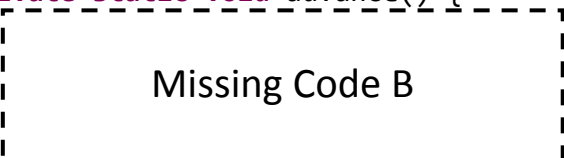
```
public static void main(String[] args) {
    numButtons = Integer.parseInt(args[0]);
    Shell shell = createShell();
    // event loop - shell.pack(), shell.open(), etc.
}

/** Create the main window of the game */
private static Shell createShell() {
    Shell shell = new Shell(Display.getDefault());
    shell.setLayout(new FillLayout(SWT.VERTICAL));

    // create the label at the top of the window
    message = new Label(shell, SWT.NONE);

    // create buttons in a loop
    for (int i = 1; i <= numButtons; i++) {
        Button b = new Button(shell, SWT.PUSH);
        b.setText(Integer.toString(i));
        b.addSelectionListener(new SelectionAdapter() {
            @Override
            public void widgetSelected(SelectionEvent e) {
                
            }
        });
    }
    advance();

    return shell;
}

/** Updates the next button to press */
private static void advance() {
    
}

private static void setGameOver() {
    gameOver = true;
    message.setText("GAME OVER");
}
}
```

א. (5 נק') השלימו את הקוד החסר במקטע המסומן ב-A בעמוד הקודם.
 רמז: על מנת שהסעיפים הבאים יעבדו כראוי, הקוד צריך להשתמש במתודה advance שאת מימושה תשלימו בהמשך.

```
public void widgetSelected(SelectionEvent e) {
```

```
    if (gameOver)
        return;
    Button source = (Button) e.getSource();
    String numAsString = source.getText();
    int num = Integer.parseInt(numAsString);
    if (num == next) {
        advance();
    } else {
        setGameOver();
    }
}
```

```
}
```

ב. (5 נק') השלימו את הקוד החסר במקטע המסומן ב-B בעמוד הקודם, כך שבכל שלב (בהתחלה ולאחר כל לחיצה) המשחק יציע למשתמש ללחוץ על כפתור אקראי אחר. היעזרו במתודה nextInt של המחלקה Random (ראו בנספח).

```
private static void advance() {
```

```
    next = new Random().nextInt(numButtons) + 1;
    message.setText("Press Button " + next);
```

```
}
```

ג. (5 נק') השלימו את הקוד החסר במקטע המסומן ב-B בעמוד הקודם, כך שבכל שלב המשתמש יתבקש ללחוץ על הכפתור הבא לפי הסדר (כלומר, ללחוץ על 1, אחר כך על 2, וכולי) באופן מעגלי. כלומר, לאחר לחיצה על הכפתור האחרון נתחיל שוב מהכפתור הראשון.

```
private static void advance() {
```

```
    next = (next % numButtons) + 1;
    message.setText("Press Button " + next);
```

```
}
```

ד. (5 נק') השלימו את הקוד החסר במקטע המסומן ב-B בעמוד הקודם, כך שבכל שלב המשתמש יתבקש ללחוץ על הכפתור הקודם לפי הסדר (כלומר, ללחוץ על 1, אחר כך על הכפתור האחרון, אח"כ על הכפתור לפני האחרון, וכולי) באופן מעגלי. כלומר, לאחר לחיצה על הכפתור הראשון נתחיל שוב מהכפתור האחרון.

```
private static void advance() {
```

```
    if (next == 0)
        next = 1;
    else
        next = (next + numButtons - 2) % numButtons + 1;

    message.setText("Press Button " + next);
```

```
}
```

ניתן להשתמש בתחית העמוד כדי להוסיף שדות ומתודות עזר נוספים למחלקה PressItGame.

שאלה 4 (20 נקודות)

התבוננו בקוד המחלקה הבאה וענו על השאלות.

```
public class B {

    private int f = 0;

    public static void main(String[] args) {
        B b1 = new B();
        B b2 = new B();
        Object b3 = b1;

        System.out.println(/***/);
    }

    @Override
    public boolean equals(Object obj) {
        f++;
        B b = (B) obj;
        return b.f == (f - 1);
    }

    public boolean eq(B b) {
        return super.equals(b);
    }
}
```

בכל אחד מהסעיפים הבאים מוחלף סימון ה- `/***/` בפונקציית ה-`main` בקטע קוד. הנכם מתבקשים לציין מהו הפלט של הרצת פונקציית ה- `main` בכל אחד מהמקרים ע"י בחירת אחת מארבע האפשרויות הנתונות והקפתה בעיגול.

שימו לב:

- בכל מקרה (שגיאה או ריצה תקינה) יש לכתוב הסבר קצר.
- ניתן להניח כי כאשר `/***/` נמחק ולא מוחלף בקוד, התכנית מתקמפלת ללא שגיאות.

סעיף א' (4 נק')

`b1.equals(null)`

`/***/` מוחלף ב-

i. יודפס `true` ii. יודפס `false` iii. תתקבל שגיאת קומפילציה iv. תתקבל שגיאה בזמן ריצה

הסבר: המימוש של `equals` שנקרא הוא המימוש הדורס של `B`. בשורה השלישית מנסים לגשת לשדה של `obj`, ומכיוון שהוא `null`, בזמן ריצה ייזרק חריג מסוג `NullPointerException`.

סעיף ב' (4 נק')

b1.equals(b2)

/**/ מוחלף ב-

i. יודפס true ii. יודפס false iii. תתקבל שגיאת קומפילציה iv. תתקבל שגיאה בזמן ריצה

הסבר: המימוש של equals שנקרא הוא המימוש הדורס של B. מימוש זה מקדם את השדה f של b1 ל-1 בעוד השדה של b2 נותר 0. לכן, ההשוואה של 0 ל 1-1 מחזירה true.

סעיף ג' (4 נק')

b3.equals(b3)

/**/ מוחלף ב-

i. יודפס true ii. יודפס false iii. תתקבל שגיאת קומפילציה iv. תתקבל שגיאה בזמן ריצה

הסבר: המימוש של equals שנקרא הוא המימוש הדורס של B (לפי הטיפוס הדינאמי). מימוש זה משווה את השדה f של b3 לעצמו פחות 1, ולכן ההשוואה בהכרח מחזירה false.

סעיף ד' (4 נק')

b2.eq(b2)

/**/ מוחלף ב-

i. יודפס true ii. יודפס false iii. תתקבל שגיאת קומפילציה iv. תתקבל שגיאה בזמן ריצה

הסבר: eq קוראת למתודת equals של super, כלומר, של Object. מתודה זו מחזירה true על שתי הפניות לאותו עצם בזיכרון, כמו במקרה הנ"ל.

סעיף ה' (4 נק')

b1.eq(b3)

/**/ מוחלף ב-

i. יודפס true ii. יודפס false iii. תתקבל שגיאת קומפילציה iv. תתקבל שגיאה בזמן ריצה

הסבר: eq מצפה לקבל כארגומנט עצם מטיפוס B אך b3 הוא מטיפוס Object, לכן הקומפילטר מדווח על שגיאה.

נספח

Class String

Methods	
Modifier and Type	Method and Description
char	charAt (int index) Returns the char value at the specified index.
boolean	contains (CharSequence s) Returns true if and only if this string contains the specified sequence of char values.
boolean	endsWith (String suffix) Tests if this string ends with the specified suffix.
boolean	equals (Object anObject) Compares this string to the specified object.
int	hashCode () Returns a hash code for this string.
int	indexOf (String str) Returns the index within this string of the first occurrence of the specified substring.
int	indexOf (String str, int fromIndex) Returns the index within this string of the first occurrence of the specified substring, starting at the specified index.
boolean	isEmpty () Returns true if, and only if, length() is 0.
int	lastIndexOf (String str) Returns the index within this string of the last occurrence of the specified substring.
int	lastIndexOf (String str, int fromIndex) Returns the index within this string of the last occurrence of the specified substring, searching backward starting at the specified index.
int	length () Returns the length of this string.
String []	split (String regex) Splits this string around matches of the given regular expression .
boolean	startsWith (String prefix) Tests if this string starts with the specified prefix.
String	substring (int beginIndex) Returns a new string that is a substring of this string.
String	substring (int beginIndex, int endIndex) Returns a new string that is a substring of this string.

Interface Iterator<E>

Methods	
Modifier and Type	Method and Description
boolean	hasNext () Returns true if the iteration has more elements.
E	next () Returns the next element in the iteration.
void	remove () Removes from the underlying collection the last element returned by this iterator (optional operation).

Interface Set<E>

Methods	
Modifier and Type	Method and Description
boolean	add (E e) Adds the specified element to this set if it is not already present. Returns true if this set did not already contain the specified element.
boolean	addAll (Collection <? extends E > c) Adds all of the elements in the specified collection to this set if they're not already present (optional operation).
void	clear () Removes all of the elements from this set (optional operation).
boolean	contains (Object o) Returns true if this set contains the specified element.
boolean	containsAll (Collection <?> c) Returns true if this set contains all of the elements of the specified collection.
boolean	equals (Object o) Compares the specified object with this set for equality.
int	hashCode () Returns the hash code value for this set.
boolean	isEmpty () Returns true if this set contains no elements.
Iterator < E >	iterator () Returns an iterator over the elements in this set.
boolean	remove (Object o) Removes the specified element from this set if it is present. Returns true if this set contained the element
boolean	removeAll (Collection <?> c) Removes from this set all of its elements that are contained in the specified collection (optional operation).
boolean	retainAll (Collection <?> c) Retains only the elements in this set that are contained in the specified collection (optional operation).
int	size () Returns the number of elements in this set (its cardinality).

Interface List<E>

Method Summary	
boolean	add (E e)
boolean	addAll (Collection <? extends E > c)
void	clear ()
boolean	contains (Object o)
E	get (int index)
int	indexOf (Object o)
Iterator < E >	iterator ()
E	remove (int index)
E	set (int index, E element)
int	size ()

Interface Map<K,V>

Method Summary	
void	clear ()
boolean	containsKey (Object key)
boolean	containsValue (Object value)
Set < Map.Entry <K,V>>	entrySet ()
V	get (Object key)
boolean	isEmpty ()
Set <K>	keySet ()
V	put (K key, V value)
V	remove (Object key)
int	size ()

Class Collections

Method Summary	
static <T extends Comparable <? super T>> void	sort (List <T> list) Sorts the specified list into ascending order, according to the <i>natural ordering</i> of its elements.
static <T> void	sort (List <T> list, Comparator <? super T> c) Sorts the specified list according to the order induced by the specified comparator.

Class Random

Constructor Summary	
Random ()	Creates a new random number generator.
Method Summary	
int	nextInt (int n) Returns a pseudorandom, uniformly distributed int value between 0 (inclusive) and the specified value (exclusive), drawn from this random number generator's sequence.

Class Object

Method Summary	
boolean	equals (Object obj) Indicates whether some other object is "equal to" this one.
Class <?>	getClass () Returns the runtime class of this Object.
int	hashCode () Returns a hash code value for the object.
String	toString () Returns a string representation of the object.

GUI

Class Button

Methods	
Modifier and Type	Method and Description
void	addSelectionListener (SelectionListener listener) Adds the listener to the collection of listeners who will be notified when the control is selected by the user, by sending it one of the messages defined in the <code>SelectionListener</code> interface.

Interface SelectionListener

Methods	
Modifier and Type	Method and Description
void	widgetDefaultSelected (SelectionEvent e) Sent when default selection occurs in the control.
void	widgetSelected (SelectionEvent e) Sent when selection occurs in the control.

Class SelectionEvent

Methods	
Modifier and Type	Method and Description
String	toString () Returns a string containing a concise, human-readable description of the receiver.
Object	getSource () The object on which the Event initially occurred.