בחינת מועד א', סמסטר א', תשס"א
7.2.2021
בחינה זו נערכה באופן מקוון.

שאלה 1 – 8 נק' (לשאלה זו יש יותר מגירסא אחת):
מה יודפס בהרצת הקוד הבא?

```
public class Test {

	public static void replaceArr(int[] arr1, int[] arr2) {
		arr1[0] = 10;
		arr2 = arr1;
	}

	public static void main(String[] args) {
		int[] arr1 = {1,2,3};
		int[] arr2 = {4,5,6};
		int[] arr3 = {1,2,3};
		int[] arr4 = arr2;
		boolean b1 = arr1 == arr3;
		replaceArr(arr1, arr2);
		boolean b2 = arr1[0] == arr3[0];
		boolean b3 = arr2 == arr4;
		System.out.print(b1 + " " + b2 + " " + b3);
	}
}
```

שאלה זו מצריכה נימוק. הסבירו כל הדפסה שמתבצעת.
בחר\י בתשובה הטובה ביותר:

1. יודפס false false false
2. יודפס false false true
3. יודפס false true false
4. יודפס false true true
5. יודפס true false false
6. יודפס true false true
7. יודפס true true false
8. יודפס true true true

גירסא נוספת של הקוד:

```
public class Test {

	public static void replaceArr(int[] arr1, int[] arr2) {
		arr1[0] = 10;
		arr2 = arr1;
	}

	public static void main(String[] args) {
		int[] arr1 = {1,2,3};
		int[] arr2 = {4,5,6};
		int[] arr3 = {1,2,3};
		int[] arr4 = arr2;
		boolean b1 = arr1 == arr3;
		replaceArr(arr1, arr2);
		boolean b2 = arr2 == arr4;
		boolean b3 = arr1[0] == arr3[0];
		System.out.print(b1 + " " + b2 + " " + b3);
	}
}
```

<u>שאלה 2:</u>

שאלות 2-4 מתייחסות לקוד הבא:

הסטודנטית שי מעוניינת לממש איטרטור אשר בהנתן אוסף (Collection) יעבור אך ורק על האיברים אשר מקיימים תנאי מסויים (Predicate). איטרטור זה ימומש במחלקה PredIterator

נשתמש במנשק הקיים Predicate המוגדר באופן הבא:

```
public interface Predicate<T>{
	boolean test(T t)
}
```

הפונקציה test מחזירה true אם הפרדיקט מתקיים עבור t, ו false אחרת.

על מנת לאפשר שימוש כמה שיותר נרחב לאיטרטור החדש, נרצה להרכיב אותו על איטרטורים אחרים באופן הבא:

```
List<Integer> myList = Arrays.asList(1,2,3,4,5);
PredIterator<Integer> iter1 = new PredIterator<>(myList.iterator(),
                              x->x %2 == 0);
while(iter1.hasNext()){
	System.out.print(iter1.next());
}
// This loop should have two iterations, the printed number are 2,4

PredIterator<Integer> iter2 = new PredIterator<>(iter1, x-> x>3);
while(iter2.hasNext()){
	System.out.println(iter2.next());
```

```
}
// This loop should a single iterator and print the number 4
```

לפניכם מימוש חלקי של המחלקה, ובו שני קטעי קוד שהוחלפו ברצף של ** ושל ??:

```
public class ********* {

	protected Iterator<K> it;
	protected ??????????? pred;

	public PredIterator(Iterator<K> it, ????????? pred){
		this.it = it;
		this.pred = pred;
	}

	public boolean hasNext() {
		//Q7
	}

	public K next() {
		//Q7
	}
}
```

כיצד ניתן להשלים את הקוד ? לפניכם מספר אופציות לכל סימון:

עבור *********

```
PredIterator<K> implements Iterator<K>    //S1

PredIterator<K> implements Iterator       //S2

PredIterator<K>                           //S3
```

עבור ???????????

```
Predicate<K> //Q1

Predicate    //Q2
```

אילו שילובים מתאימים לשלד המימוש ולדוגמת השימוש בקוד ולשלד הקוד?

1. רק S1 + Q1
2. רק S1 + Q2
3. רק S2 + Q1
4. רק S2 + Q2
5. רק S3 + Q1
6. רק S3 + Q2
7. יש שני שילובים מתאימים

8. יש שלושה שילובים מתאימים.

שאלה 3:

נמשיך במימוש השירותים hasNext ו next. לפניכם מימוש של הסטודנטית שי. הניחו כי הטיפוסים שהייתם צריכים להגדיר בשאלה הקודמת הוגדרו נכון ובהתאמה למימוש המסופק.

```
public boolean hasNext() {
	return it.hasNext();
}

public K next() {
	while (it.hasNext()){
		K res = it.next();
		if (pred.test(res)){
			return res;
		}
	}
   return null;
}
```

מה יקרה בעת הרצת הלולאה הראשונה משאלה 2 ~~10~~ (הלולאה על האיטרטור iter1)?

1. הלולאה תרוץ במצופה, תדפיס את האיברים 2 ו 4 ואז תעצור.
2. במהלך ריצת הלולאה תיזרק שגיאת זמן ריצה בפונק' next.
3. הלולאה תרוץ 3 פעמים ותדפיס את האיברים 2, 4 ו null.
4. הלולאה תרוץ רק פעם אחת ותדפיס את המספר 2.
5. הלולאה תרוץ רק פעם אחת ותדפיס את המספר 4.
6. הלולאה תרוץ 5 פעמים ותדפיס את האיברים null, 2, null, 4, null.

שאלה 4 (6 נק'):

הסטודנטית שי מעוניינת לממש מחלקה חדשה: MultiPredIterator. מחלקה זו מקבלת איטרטור ורשימה של פרדיקטים, ומאתחלת איטרטור אשר עובר על כל האיברים אשר מקיימים את כל הפרדיקטים.

זוהי דוגמא לאופן השימוש:

```
List<Predicate<Integer>> preds = Arrays.asList(x-> x>3, x-> x%2 == 0);
MultiPredIterator<Integer> iter3 = new MultiPredIterator<>(iter1, preds);
```

האיטרטור iter3 יחזיר את כל האיברים עליהם עובר iter1 אשר עומדים בכל הפרדיקטים שהוגדרו ב preds.

שי מעוניינת לעשות שימוש חוזר במחלקה PredIterator ע"י ירושה (ניתן להניח שאם בשאלה 3 המימוש של next ו hasNext לא היה תקין, שי תיקנה את המימוש). כמו מתכנתות רבות, שי ניגשה לאינטרנט, חיפשה קוד דומה וביצעה התאמות קלות.

```
class MultiPredIterator<K> extends PredIterator<K>{
      public MultiPredIterator(Iterator<K> it,
                                    List<Predicate<K>> pred) {
            super(it, x->{
                  for (Predicate<K> p: pred) {
                        if (p.test(x)) {return true;}
                  }
                  return false;
            });
      }
}
```

(הערה – בקוד זה מתבצעת קריאה ל super עם פונק' למבדא. אין בעיה בקריאות מהסוג הזה, ואם קיימת שגיאת קומפילציה בקוד הזה היא נובעת רק מבעיית טיפוסים.)

בחר\י בתשובה הטובה ביותר?

שאלה זו מצריכה נימוק. הסבירו בקצרה את רעיון המימוש. במידה שהקוד לא עובד כנדרש, הסבירו מדוע.

א. הקוד לא מתקמפל כיוון שהטיפוס שמועבר כפרמטר השני ל super לא תואם לטיפוס המצופה.

ב. הקוד מתקמפל ורץ ועובד כנדרש, ללא צורך בעדכון של next ו hasNext.

ג. הקוד מתקמפל אך זורק זמן שגיאה בזמן ריצה.

ד. הקוד מתקמפל ורץ, אך לא יחזיר את האיברים הנכונים. על מנת שיחזיר את האיברים הנכונים, מספיק לבצע תיקון בקוד של הבנאי.

ה. הקוד מתקמפל ורץ, אך לא יחזיר את האיברים הנכונים. על מנת שיחזיר את האיברים הנכונים, צריך לדרוס את next ו hasNext.

ו. לא ניתן לממש את MultiPredIterator ע"י ירושה מ PredIterator. אצל חלק מהסטודנטים המסיח הזה הופיע הפוך בגלל בעיית עברית\אנגלית (PI לא יכול לרשת מ MPI). בשני המקרים, הטענה לא נכונה.

שאלה 5 (8 נקודות):

לפניכם שלוש טענות הקשורות לחוזים: טענות 1+2 מתייחסות למחלקות Sub ו Base, וטענה 3 היא כללית.

```
public class Base{
      /*
       * @pre: i % 2 == 0
       * @post $ret > -10
       */
      public int func(int i) {
```

```
            return Math.abs(i)/2;
      }
}

public class Sub extends Base{
      /*
       * @pre: i % 2 == 0 and i >= 0
       * @post $ret > -10
       */
      public int func(int i) {
            return super.func(i);
      }
}
```

טענה 1: המימוש של הפונקציה func של Base מקיים את החוזה שלה.

טענה 2: החוזה של הפונקציה func של Sub הוא חוקי על פי חוקי הירושה.

טענה 3: עבור מחלקה X כלשהי, שמורה I (invariant) כלשהי ופונקציה f כלשהי ב X נגדיר את הטענה הבאה:

"אם השמורה I מתקיימת לפני הפעלת f אז היא מתקיימת גם לאחר הפעלת f".

אם טענה זו מתקיימת לכל פונקציה f ב X, אזי השמורה I מתקיימת עבור המחלקה X.

שאלה זו מצריכה נימוק. לכל טענה, נמקו בקצרה את תשובתכם לגבי הנכונות שלה.

תשובה: רק טענה 1 נכונה.

שאלה 6 : (לשאלה זו יש יותר מגירסא אחת)

מה יקרה בהרצת הקוד הבא?

```
public class ExceptionsTest {
	public static Object func(int i) throws Exception{
		if (i %2 == 0){
			throw new Exception();
		}
		if (i %3 == 0){
			throw new NullPointerException();
		}
		if (i %5 == 0){
			return new Exception();
		}
		return 2;
	}

	public static void main(String[] args){
		int sum = 0;
		List<Integer> lst = Arrays.asList(1,2,3,5);
		try{
			for (int num : lst){
				try{
					func(num);
					sum+=1;
				}
				catch(Exception exp){
					sum += 1;
				}
			}
		}
		finally{
			sum += 1;
		}
		System.out.println(sum);
	}
}
```

שאלה זו מצריכה נימוק. בכל איטרציה, הסבירו בקצרה מה קורה בקוד.

בחר\י בתשובה הטובה ביותר:

א. תיזרק שגיאת זמן ריצה ולא יודפס כלום.

ב. יודפס 5.

ג. מסיחים מ 2 עד 11.

גירסא שניה:

```
public class ExceptionsTest {
	public static Object func(int i) throws Exception{
		if (i %2 == 0){
			throw new Exception();
		}
		if (i %3 == 0){
			throw new NullPointerException();
		}
		if (i %5 == 0){
			return new Exception();
		}
		return 2;
	}

	public static void main(String[] args){
		int sum = 0;
		List<Integer> lst = Arrays.asList(1,2,3,5);
		try{
			for (int num : lst){
				try{
					func(num);
					sum+=1;
				}
				catch(Exception exp){
					sum += num;
				}
			}
		}
		finally{
			sum += 1;
		}
		System.out.println(sum);
	}
}
```

גירסא שלישית:

```java
public class ExceptionsTest {
	public static Object func(int i) throws Exception{
		if (i %2 == 0){
			return new Exception();
		}
		if (i %3 == 0){
			throw new NullPointerException();
		}
		if (i %5 == 0){
			throw new Exception();
		}
		return 2;
	}

	public static void main(String[] args){
		int sum = 0;
		List<Integer> lst = Arrays.asList(1,2,3,5);
		try{
			for (int num: lst){
				try{
					func(num);
					sum+=num;
				}
				catch(Exception exp){
					sum += 1;
				}
			}
		}
		finally{
			sum += 1;
		}
		System.out.println(sum);
	}
}
```

שאלה 7:
לפניכם פסאודו קוד העושה שימוש בזרמים (Streams). הפונקציות המופיעות בשורה השניה לא אמיתיות, ומחליפות פעולות ביניים (intermediate) ופעולות סופניות (terminal) אמיתיות על הזרם.

```
Stream<Integer> s = ???
s.someIntermediate().otherIntermediate().terminal();
System.out.println("done!");
```

טענה 1: אם נסיר את הקריאה ל terminal, הקוד לא יגיע לשורה האחרונה (ההדפסה).

טענה 2: ניתן לייצר זרם אינסופים של Integer-ים כך שאף מספר לא יחזור על עצמו פעמיים.

טענה 3: נניח כי s הוא זרם אינסופי ונפעיל עליו את הפעולות הבאות:

```
s.filter(???).forEach(System.out::println);
```

ניתן לממש את הפרדיקט (פרדיקט) שמקבל ה filter כך שריצת השורה הזו תסתיים והקוד ימשיך לשורה הבאה.

שאלה זו מצריכה נימוק. לכל טענה, הסבירו מדוע היא נכונה או לא נכונה.

תשובה – כל הטענות לא נכונות.

שאלה 8: (לשאלה זו יש יותר מגירסא אחת)

לפניכם קוד שבו נמצאים בהערה שלושה מימושים לפונקציה func של המחלקה Base.

```
public class Base{
	/*
	public Object func(String str) { //impl 1
		//implementation here
	} */
	/*
	public String func(String str) throws Exception{ //impl 2
		//implementation here
	}
	*/
	/*
	public String func(Object str){ //impl 3
		//implementation here
	}
	*/
}

public class Sub extends Base{
	public String func(String str) { return null; }

	public static void main(String[] args){
		Base a = new Sub();
		a.func("abc");
	}
}
```

השאלה מתייחסת לקומפילציה של המחלקה Sub ושל הפונקציה main. הניחו כי Base כשלעצמה מתקמפלת בכל אחת מהאופציות.

הערה – לכל מימוש של func שתבחנו, הניחו כי המימושים האחרים נמצאים בהערה.

לפניכם שלוש טענות על המימושים של func בתוך Base.

טענה 1: אם נוציא את מימוש 1 של func מההערה - הקוד יתקמפל.

טענה 2: אם נוציא את מימוש 2 של func מההערה - הקוד יתקמפל.

טענה 3: אם נוציא את מימוש 3 של func מההערה - הקוד יתקמפל.

בחר\י בתשובה הטובה ביותר

תשובה: רק טענות 3+1 נכונות (טענה 2 לא נכונה. אם נוציא את imp2 מההערה, הדריסה היא אמנם חוקית, אבל ה main לא יתקמפל בגלל func של base מצהירה על חריג שלא טופס\נתפס.

גירסא נוספת של הקוד:

```java
public class Base{
	/*
	public Object func(String str) { //impl 1
		//implementation here
	} */
	/*
	public String func(Object str) throws Exception{ //impl 2
		//implementation here
	}
	*/
	/*
	public String func(Object str){ //impl 3
		//implementation here
	}
	*/
}

public class Sub extends Base{
	public String func(String str) { return null; }

	public static void main(String[] args){
		Base a = new Sub();
		a.func("abc");
	}
}
```

שאלה 9:

בשאלה זו נכתב במקור BufferedReader במקום BufferedWriter. השאלה תוקנה בזמן הבחינה. מכיוון שכל ההתייחסות היא לכתיבה ולא לקריאה, הניסוח המקורי של השאלה בעצם הופך את טענות 1+2 לטריוויאליות, כיוון ש BR לא משתמש לכתיבה.

לפניכם מספר טענות על ה BufferedWriter. להזכירכם, באחד מתרגילי הבית מימשתם את המחלקה MyBufferedWriter אשר דומה בהתנהגותו ל BufferedWriter.

טענה 1: לא ניתן לכתוב לקובץ ללא שימוש ב BufferedWriter.

טענה 2: ה BufferedWriter כותב את אותה כמות התווים בכל כתיבה לקובץ.

טענה 3: נגדיר

```java
FileWriter fw = new FileWriter("c:/my_file.txt");
BufferedWriter bw = new BufferedWriter(fw);
```

אם נרצה להקטין את מספר פעולות הכתיבה (write) שמבצע ה fw, עלינו להקטין את גודל ה buffer של bw (הניחו שלא מתבצעים שינויים נוספים מלבד שינוי גודל ה buffer)

שאלה זו מצריכה נימוק. לכל טענה שסומנה כלא נכונה, נמקו ברצה מדוע אינה מתקיימת.

בחר\י בתשובה הטובה ביותר:

א. רק טענה 1 נכונה.
ב. רק טענה 2 נכונה.
ג. רק טענה 3 נכונה.
ד. רק טענות 1+2 נכונות.
ה. רק טענות 1+3 נכונות.
ו. רק טענות 1+2 נכונות.
ז. כל הטענות נכונות.
ח. כל הטענות לא נכונות.

שאלה 10:

לפניכם מימוש של שלוש פונקציות:

```
public class Test {
	public void func1(List<? super Exception> lst){
			lst.add(new IOException());
	}

	public void func2(List<Exception> lst){
			lst.add(new IOException());
	}

	public void func3(List<?> lst){
			lst.add(new Object());
	}
}
```

בחר\י בתשובה הטובה ביותר:

א. רק פונקציה 1 מתקמפלת.
ב. רק פונקציה 2 מתקמפלת.
ג. רק פונקציה 3 מתקמפלת.
ד. רק פונקציות 1+2 מתקמפלות.
ה. רק פונקציות 1+3 מתקמפלות.
ו. רק פונקציות 2+3 מתקמפלות.
ז. כל הפונקציות מתקמפלות.
ח. כל הפונקציות לא מתקמפלות.

(גירסא 2 של הקוד)

```
public class Test {
	public void func1(List<?> lst){
			lst.add(new Object());
	}


	public void func2(List<? extends Exception> lst){
			lst.add(new IOException());
	}

	public void func3(List<Exception> lst){
		lst.add(new IOException());
	}
}
```

(גירסא 3 של הקוד)

```
public class Test {
	public void func1(List<? extends Exception> lst){
			lst.add(new IOException());
	}

	public void func2(List<Object> lst){
		lst.add(new Object());
	}


	public void func3(List<? extends IOException> lst){
		lst.add(new Exception());
	}

}
```

<u>שאלה 11 (לשאלה זו יש יותר מגירסא אחת)</u>
לפניכם מספר טענות על שגיאות זמן ריצה וניהול הזכרון ב Java.

טענה 1: אם נוסיף אובייקטים לרשימה בלולאה אינסופית, נקבל שגיאת StackOverflowError.

טענה 2: שדות מופע (instance members) מטיפוסים פרימיטיביים נשמרים על המחסנית.

טענה 3: שגיאה מסוג unchecked ניתן לתפוס באמצעות בלוק try/catch

בחר\י בתשובה הטובה ביותר:

תשובה – רק טענה 3 נכונה.

שאלה 12 (לשאלה זו יש יותר מגירסא אחת):

```
public abstract class Abst{

	public abstract int getFirst();

	private int getSecond(){ //*
		return 3;
	}

	public void printRes(){
		System.out.print(getFirst() + getSecond()); //**
	}
}

public class B extends Abst{
	public int getFirst(){
		return 5 + getSecond();
	}

	private int getSecond(){ return 6; }
}

public class C extends B{
	public int getFirst(){
		return 2*super.getFirst();
	}

	public int getSecond(){ return 2; }

	public static void main(String[] args){
		Abst a = new C(); //***
		a.printRes();
	}
}
```

מה יקרה בהרצת התוכנית C?

1. יש שגיאת קומפילציה בשורה * בגלל הגדרת פונקציה private במחלקה אבסטרקטית.
2. יש שגיאת קומפילציה בשורה ** בגלל קריאה לפונקציה שלא מומשה.
3. יש שגיאת קומפילציה בשורה *** בגלל שמחלקה אבסטרקטית לא יכולה להיות טיפוס סטטי של משתנה.
4. התוכנית תרוץ ויודפס 25

5. התוכנית תרוץ ויודפס 23
6. התוכנית תרוץ ויודפס 21
7. התוכנית תרוץ ויודפס 19
8. התוכנית תרוץ ויודפס 17
9. התוכנית תרוץ ויודפס 15
10. התוכנית תרוץ ויודפס 13

גירסא שניה של הקוד:

```
public abstract class Abst{

	public abstract int getFirst();

	private int getSecond(){ //*
		return 3;
	}

	public void printRes(){
		System.out.print(getFirst() + getSecond()); //**
	}
}

public class B extends Abst{
	public int getFirst(){
		return 3 + getSecond();
	}

	private int getSecond(){ return 6; }
}

public class C extends B{
	public int getFirst(){
		return 2*super.getFirst();
	}

	public int getSecond(){ return 2; }

	public static void main(String[] args){
		Abst a = new C(); //***
		a.printRes();
	}
}
```

שאלה 13:

בשאלה זו נתונות לכם 3 הגדרות אפשריות עבור המחלקה הפנימית A המוגדרת בתוך המחלקה HelloWorld.

```
public class HelloWorld{

	protected int w = 5;

	//public class A {  //op1
	//public static class A { //op2
	//public abstract class A { //op3
		public int i = 0;

		public A(){
		    this(1);
		}
		public A(int a){
		    this(a,2*a);
		    System.out.print(a);
		}
		public A(int a, int b){
		    setI();
		    System.out.print(a+b+i); //*
		}

		public void setI() {
			HelloWorld h =  new HelloWorld();
			this.i = h.w;   //*
		}
	    }

	public static void main(String[] args){
		A a = new A(); //*
	}
}
```

נרצה לבדוק עבור כל הגדרה אפשרית של A אם הקוד מתקמפל. המקומות הבעייתים שבהם עשויה להיות שגיאת קומפילציה סומנו ב *.

לפניכם 3 טענות:
טענה 1: אם נוציא מהערה את op1 התוכנית תתקמפל ותרוץ.
טענה 2: אם נוציא מהערה את op2 התוכנית תתקמפל ותרוץ.
טענה 3: אם נוציא מהערה את op3 התוכנית תתקמפל ותרוץ.

בחר\י בתשובה הטובה ביותר:

1. כל הטענות לא נכונות (כלומר, הקוד לא יתקמפל עבור כל אחת מהאופציות).
2. טענה 1, ויודפס 81
3. טענה 2, ויודפס 81
4. טענה 3, ויודפס 76
5. טענות 1+2, ויודפס 81

6. טענות 3+1, ויודפס 76
7. טענות 3+2, ויודפס 81
8. כל הטענות נכונות, ויודפס 76

שאלה 14:
מה יקרה בהרצת התוכנית הבאה?

```
public class Test{
	private static A a1 = new A();
	private A a2 = new A();

	public static void main(String[] args){
		Test t1 = new Test();
		System.out.print(t1.a2.getValue());
		Test t2 = new Test();
		Test t3 = new Test();
		System.out.print(t1.a2.getValue());
	}
}


public class A{
	private static int cnt;
	private int i;

	public A(){
		cnt++;		//*
		this.i = cnt;
	}

	public int getValue() {return this.i + cnt;}
}
```

א. שגיאת קומפילציה בשורה המסומנת ב *
ב. התוכנית תרוץ ויודפס 44
ג. התוכנית תרוץ ויודפס 46
ד. התוכנית תרוץ ויודפס 48
ה. התוכנית תרוץ ויודפס 22
ו. התוכנית תרוץ ויודפס 24
ז. התוכנית תרוץ ויודפס 26
ח. התוכנית תרוץ ויודפס 28

שאלה 15:

לפניכם מספר טענות הנוגעות ל Comparator ו Comparable:

בחר\י בתשובה הטובה ביותר:

א. ניתן לממש Comparator רק למחלקה שממשת את Comparable.
ב. לכל מחלקה ניתן לממש Comparator אחד בלבד.
ג. המשמעות של ההגדרה X implements Comparable <Y>

היא שניתן להשוות אובייקטים מטיפוס X עם אובייקטים מטיפוס Y.
ד. מלבד תשובה זו כל התשובות לא נכונות.
ה. מלבד תשובה זו יש יותר מתשובה נכונה אחת.

שאלה 16 (לשאלה זו קיימות מספר גירסאות):

נתון הקוד הבא:

```
A a = new A();
Object clone= a.clone();
```

בחר\י בתשובה הטובה ביותר:
א. הקוד מתקמפל לכל מימוש של A, אך יתכנו שגיאות בזמן ריצה.

ב. הקוד יתקמפל רק אם A ממשת את cloneable.

ג. אם A רוצה לממש את clone, היא חייבת לעשות שימוש במימוש שנורש מ Object ע"י הפעלת (super.clone)

ד. מלבד תשובה זו כל התשובות לא נכונות.

ה. מלבד תשובה זו, יש יותר מתשובה נכונה אחת.

שאלה 17:

```
public class Grandma{
	public Grandma(int i) {}
}

public class Mom extends Grandma{
	public Mom(int i1, int i2) {
		//super();  // m1
	}
}

public class Child extends Mom{
	public Child(String str) {
		//this(1, 2);	// c1
		//super(1);	// c2
	};

	public Child(int i1, int i2) {this("abc"); };
}
```

בקוד הנתון יש שלוש השלמות אפשריות לבנאים של Mom ושל Child. עליכם לבחון כל השלמה אפשרית בנפרד. שימו לב שעבור הקוד של Child, עליכם להניח שהבנאי שנתון עבור Mom ממומש כך שהוא מתקמפל.

טענה 1: אם נוציא את שורה m1 מההערה, הקוד של Mom יתקמפל.
טענה 2: אם נוציא את שורה c1 מההערה, הקוד של Child יתקמפל (בהנחה שגם Mom מתקמפלת).
טענה 3: אם נוציא את שורה c2 מההערה, הקוד של Child יתקמפל (בהנחה שגם Mom מתקמפלת).

בחר\י בתשובה הטובה ביותר:

א. רק טענה 1 נכונה.
ב. רק טענה 2 נכונה.
ג. רק טענה 3 נכונה.
ד. רק טענות 1+2 נכונות.
ה. רק טענות 1+3 נכונות.
ו. רק טענות 2+3 נכונות.
ז. כל הטענות נכונות.
ח. כל הטענות לא נכונות.

שאלה 18:

```
public class Point implements Comparable<Point>{
	private int i,j;

	public Point(int i, int j) {
		this.i = i;
		this.j = j;
	}

	public boolean equals(Object other) {
		Point oP = (Point)other;
		return this.i == oP.i && this.j == oP.j;
	}

	public int hashCode() {return i;}

	@Override
	public int compareTo(Point o) {
		return Integer.compare(this.i,o.i)
				* Integer.compare(this.j, o.j);
	}

	public static void test(Set<Point> mySet) {
		mySet.add(new Point(1,2));
		mySet.add(new Point(1,2));
		mySet.add(new Point(1,3));
		mySet.add(new Point(2,1));
		System.out.print(mySet.size() +" ");
	}

	public static void main(String[] args) {
		Set<Point> myHashSet = new HashSet<>();
		Set<Point> myTreeSet = new TreeSet<>();
		test(myHashSet);
		test(myTreeSet);
	}
}
```

מה יודפס בהרצת התוכנית main? (הערה – ניתן להניח בשאלה זו שב TreeSet כל איבר שמוכנס מושווה לכל האיברים שכבר נמצאים ב set. זה לא מדוייק, אבל במקרה הספציפי המובא בשאלה הזו ההנחה הזו מתאימה).

א. 1 1

ב. 1 2

ג. 1 3

ד. 2 1

ה. 2 2

ו. 2 3

ז. 3 1

ח. 3 2

ט. 3 3

י. 4 1

יא. 4 2

יב. 4 3