



מדריך JAVA

תוכן עניינים

2	מבוא לתכנות ב-JAVA
5	סינטקס וצורת כתיבה ב-JAVA
8	משתנים
10	קלט מהמשתמש
12	בונוס - הגרלת מספר רנדומלי
13	אובייקטים
15	תכנות מונחה עצמים
17	פולימורפיזם
20	מחלקה אבסטרקטית
22	בניית ממשק - interface



מבוא לתכנות בJAVA

שפת JAVA זוהי שפת תכנות אשר נחשבת לשפת על בעולם ההייטק, וזאת מהסיבה שהיא מסוגלת לבצע המון פעולות בניגוד לשפות פיתוח אחרות. בנוסף לכל הפעולות הרב-גוניות שהיא מסוגלת לבצע, היא גם מסוגלת להתלבש על כל פלטפורמה ועל כל מערכת. היא מאוד פופולרית ותופסת תאוצה אדירה בשנים האחרונות. סביבת העבודה שלנו תהיה bluej. זוהי תוכנה חינמית אשר מתאימה למתחילים ומציגה בצורה מאוד ברורה את כל עולם הג'אווה, גם את מאחורי הקלעים. לכן, אנא לפני תחילת העבודה הורידו את התוכנה בלחיצה כאן ונתחיל לפתח בג'אווה ביחד דרך מדריך זה. למעשה, בסרטון הנלווה מצורף הסבר ההורדה, מהי bluej ואיך להתקין. הביטו בו. לאחר שהתקנתם את התוכנה, נתחיל להבין את הפיתוח הראשוני בג'אווה ואיך מציגים הודעה למשתמש בצורה הפשוטה ביותר והראשונית ביותר: כל מסמך קוד בג'אווה נקרא class. כמו שיש לנו לכל מסמך WORD את הDOC שלו, כל מסמך טקסט את הTXT, כל דף באתר את הHTML שלו - במקרה של java חובה לפתוח class חדש. בלחיצה ימנית נוכל לבצע זאת, יש שם class שונים ומגוונים - אנו נבחר תחילה בclass הבסיסי והראשוני, שימו לב להדרכה בסרטון.

מיד כל class ייפתח בשורה הבאה (בהנחה ששם class שבחרנו הוא first. ניתן לבחור כל שם שעולה על דעתכם).

```
public class first{  
}
```



בתוך class אנו נזין קטעי קוד שונים, או מה שנקרא 'פונקציות' (ראה ערך 'פונקציות' במדריך הלוגיקה). הפונקציה הראשית שחייבת להיות בclass הראשי שלי נקראת main. גם כאן, בלי להסביר מה המשמעות בהתחלה, אבקש מכם לכתוב אותה.

```
public static void main(){  
}
```

כך שבסופו של דבר ה class ייראה כך:

```
public class first{  
    public static void main(){  
    }  
}
```

לסיום, נלמד את הפקודה הראשונית בג'אווה. פקודת הדפסה למשתמש. כאשר נכתוב בתוך פונקציה main הראשית (כשמה כן היא) נוכל להציג למשתמש מסרים והוראות. רק דרכה. במידה ונכתוב בתוכה את הקוד הבא: `System.out.print('Hello!');` יוצג בפני המשתמש המילה Hello.

שתי הערות:

1. ניתן כמובן לשנות בתוך הסוגריים למספרים, פעולות מתמטיות וחשוביות. לא חובה רק טקסטים.

2. נא להקפיד על אותיות גדולות וקטנות וקטנה.



לאחר שסיימנו את קטע הקוד הנ"ל, אבקש מכם ללחוץ על כפתור הCompile. כפתור זה, הנמצא בצד שמאל למעלה, ממיר את הקוד שלנו לשפה בינארית וכך נבדקת שהקוד תקין ואין שגיאות. אם יש שגיאה כלשהי, כנראה שטעיתם בדרך או שכחתם משהו... לסיום, כדי להריץ, נחזור למסך הראשי, נקיש מקש ימני על קובץ הclass ונבחר בפונקציה main. תיפתח בפנינו חלונית חדשה עם המסר המודפס hello.

בשיעור הבא, נלמד איך לשמור משתנים בג'אווה (ראו ערך שיעור 'משתנים' במדריך הלוגיקה) וכן גם לאט לאט נבין מה אומרים כל המילים שטרם הסברתי (public,static,void ועוד)...





סינטקס וצורת כתיבה בJAVA

JAVA נכתוב תמיד בקובץ הCLASS, כפי שלמדנו קודם לכן, בפרק המבוא. כעת, נפרט על דגשים מיוחדים כמו הקומפלייר ('מהדר' בעברית), נקודה פסיק בסוף כל משפט ומה הרציונל שעומד מאחוריה, על טיפוסים, על הרשאות, סוגי CLASSES וכדומה...

- **הכתיבה בתוך הCLASS** - בתוך קובץ הJAVA, CLASS, תמיד נכתוב ראשית את השם שלו: `public class TAL{}` ולאחר מכן, בפנים ניצור פונקציות ראשיות אשר כל אחת מהן תייצג אלגוריתם או סדרת הוראות ופעולות לעשייה. נקפיד, על מנת להציג את המידע שלנו, נשתמש בפונקצייה שעליה פירטנו בפרק המבוא: `public static void main(){}` רק פונקציה זו מביאה לנו את האפשרות להציג את כל המידע שאנו רוצים להציג. כלומר, אם תנסו לבצע פלט למשל, כמו השורה הבאה שתציג את המילה 'TAL': `System.out.println();`, הJAVA תבקש מכם את הCONSOLE. משמע, לכתוב את השורה הנ"ל בתוך הפונקציה שיוצרת את הCONSOLE...
- **הCOMPILER** - בסופו של דבר, כל מה שנכתוב הוא טקסט. עד שהJAVA לא תמיר אותו לשפה בינארית - 0,1. איך היא עושה זאת? כיצד היא לוקחת את מה שכתבנו, כמו המילה 'BLUE' וממירה אותו לכחול באמת? באמצעות כפתור הCOMPILE אשר ממיר את כל הקוד שכתבנו בטקסט לשפה הבינארית. כהמלצה, ליחצו על כפתור הCOMPILE לאחר כל הוראה ושורת קוד כחלק מעבודה נכונה. הקומפלייר ('מהדר' בעברית) יכתוב לכם על כל שגיאה והיכן היא נמצאת...



- **נקודה פסיק בסוף כל שורת קוד** - כפי שאמרנו, הכל טקסט. והרובוט של JAVA ממש קורא שורה שורה, קוד אחרי קוד והוראה אחרי הוראה. עולה השאלה - כיצד הוא יוכל להבדיל בין ההוראות? מתי יידע מתי מסתיימת הוראה אחת, שעליו לבצע ולהתחיל הוראה אחרת? בסופו של דבר, צריך להבין - הרובוט קורא את כל שורות ה-JAVA בזו אחר זו כחטיבה אחת... ולכן, עלינו לבצע נקודה פסיק בסוף כל הוראה ופקודה. אם אנו רוצים לצבוע את ה-CONSOLE בכחול - בסוף נקודה פסיק. אם אנו רוצים להציג משהו למשתמש שלנו - נקודה פסיק בסוף! לא לשכוח, אחרת ה-COMPILER יתריע על כך.
- **הרשאות** - ב-JAVA לכל קובץ, לכל משתנה, לכל פונקציה, לכל אלגוריתם - בקיצור, להכל (!) יש הרשאה. מי יכול להביט בקוד? מי יכול לשנות אותו? רק מי שעובד באפליקציה הנוכחית או אולי גם אנשים חיצוניים? כן, זה אפשרי. ישנם מספר סוגים. במדריך זה, אדון כעת ב-**public** שהוא אפשרות ציבורית - כולם חשופים לשנות את הקוד. וכן גם, **private** - שזה פרטי. ההפך ממנו, לא ניתן לגשת לקוד. בהמשך, בחלק של הפולימורפיזם וההורשות נדבר גם על **protected**. הרשאה נוספת.
- **החזרות** - לכל פונקציה יש אפשרות להחזיר מידע כלשהו או לא. מה הכוונה? נניח, שאנחנו קוראים לפונקציה כלשהי מתוך ה-CLASS הראשי שלנו (ראה ערך, מדריך הלוגיקה - פונקציות). לעיתים, נרצה שהיא תחזיר עימה מספר כלשהו (כמו למשל, פונקציה צדדית שמחשבת ממוצע. אנו שולחים אליה מספרים והיא מחזירה אלינו תשובה - ממוצע). לעיתים, לא נרצה שהיא תחזיר דבר אלא תבצע פקודה כלשהי. לכן, יש לנו אפשרות שנקראת **VOID**.
- **סוגי משתנים** - בפרק המשתנים נדון בנושא זה. בגדול, ב-JAVA אנו צריכים להגדיר לפני כל משתנה שאנו יוצרים מה הסוג ('הטיפוס') שלו... האם הוא מספר, טקסט או משהו אחר? די זהה למשתני טבלאות ב-SQL... ודי שונה ממשתני JS...



- **STATIC** - ישנם סוג משתנה נוסף בשם STATIC. הפעם לא מדובר על 'טיפוס', אלא על השאלה האם המשתנה משותף עם CLASSים אחרים או רק בCLASS הנוכחי? למשל, אם נרצה שיהיה משתנה בשם COUNT שהוא מונה וסופר מספרים. אבל אם נקרא לו מתוך CLASS אחר, עדיין נוכל לשנות את המספר... זהו משתנה סטטי! במידה ולא נעניק למשתנה שלנו את האפשרות הזו, פשוט לא נוכל לשנות אותו מCLASS אחר... והוא יישאר תקוע עם אותו המספר שהוגדר מלכתחילה... עדיין, זה לא לנצח. כלומר, אם נצא וניכנס לתוכנה, המשתנה יתאפס חזרה למצב ההתחלתי.



משתנים

בשיעור זה אסביר על משתנים בג'אווה. ראשית, אמליץ לקרוא ולהבין לעומק את נושא המשתנים דרך מדריך הלוגיקה.

שימו לב, בסרטון יש גלישה כבר לשיעור מס' 3 במדריך זה (אודות קלט משתמש). אך נתחיל דבר-דבר ואסביר מהם המשתנים שיש לנו בשפה.

למעשה, בכל יצירה של משתנה חדש בג'אווה אנו מגדירים לפני שם המשתנה שבחרנו עבורו (יכול להיות כל שם שנרצה) את הטיפוס שלו. כלומר, מה הסוג של המשתנה? האם מספר? האם טקסט? האם בוליאני? וכדומה... כך, יוצא שלמשל אם נרצה להגדיר מספר נהיה חייבים לרשום לפני שם המשתנה num את המילה `int`.

ובתוך פונקציית הmain הראשית שלנו נצטרך להוסיף שורה של יצירת המשתנה. בתוך שורת ההדפסה נבצע את המספר ועוד עצמו.

```
public class first{  
    int num = 5;  
    public static void main(){  
        System.out.print(num+num)  
    }  
}
```

בדוגמא זו, יוצג עבור המשתמש המספר 10.



אילו עוד סוגי משתנים יש לנו בג'אווה?

- מחרוזת וטקסטים - String
- מספרים שלמים - int
- שני סוגים שימושיים פחות, הדומים ל int-אך בעלי טווח קטן יותר. מועילים בדרך כלל במצבים בהם יש צורך לחסוך בזיכרון - byte / short
- אותיות ותווים - char
- מספרים עם נקודה עשרונית. לדוגמה: 1.5, 3.278 - float
- אופרטור בוליאני, כלומר - משתנה המסוגל להכיל 'אמת' או 'שקר' בלבד - Boolean

בנוסף לעיל, יש צורת כתיבה מסודרת בג'אווה. לפני כל משתנה שאנו מגדירים נצטרך להוסיף את הגישה אליו אנו נותנים. ניתן להגדיר לפני המשתנה את הגישה **public** שהוא משתנה ציבורי, כל הclassים הנוספים שיהיו בהמשך יוכלו לדבר עם המשתנה הזה ולשנות אותו. ניתן להגדיר לפני המשתנה את הגישה **private** שהוא משתנה פרטי, רק הclassים הזה הנוכחי יוכל לדבר עם המשתנה הזה ולשנות אותו.

הערה: נציין את סוג הטיפוס ואת הגישה אותו אנו נותנים רק ברגע שאנו יוצרים את המשתנה. בכל פעם שאנו רק קוראים לו, אחרי שהוא כבר קיים במערכת (על מנת לעשות לו השמה או לשנות אותו תוך כדי תנועה) יש צורך רק לרשום את השם של המשתנה.





קלט מהמשתמש

על מנת לשדרג את הקוד שלנו ולהתממשק דרכו עם המשתמש, נוכל למשוך קטעי קוד שכתבו עבורינו מתכנתי ג'אווה. קטעי הקוד הללו שנמשוך לתוך הקוד שלנו, הם קטעי קוד שלא חלק מהג'אווה הבסיסית אלא צריך לבצע import (יבוא) שלהם לclass שלנו. הדבר יצור לכתיבה דרך המקלדת ממש בתוך consolen שלנו. שימו לב גם לסרטוני ההסבר (גם בשיעור 2).

ראשית, מעל הclass במעלה הדף נוסיף ייבוא. ניתן לייבא class שונים וחיצוניים. כעת, **נלמד על Scanner.**

```
import java.util.Scanner;
```

בשלב זה עדיין לא נעסוק באובייקטים ובבנאים (Constructors), ובשורות הבאות נשתמש בהם מבלי להבין לגמרי את משמעותם. בתוך מתודת ה-main נכתוב את השורות הבאות:

```
Scanner s = new Scanner(System.in);
```

שורה זו מאתחלת אובייקט מטיפוס Scanner בשם s, המקבל כפרמטר את זרם הקלט.

כאמור - בשלב זה עדיין לא נמצאים בידינו הכלים להבין שורה זו.

```
int num = s.nextInt();
```



למעשה, אחרי שייבאנו ואחרי שהגדרנו שהמשתנה s הוא מסוג טיפוס Scanner, ניתן ליצור את ההקלדה בתוך גוף הconsole. כל שנצטרך לעשות הוא להוסיף את השורה הנ"ל, להגדיר משתנה int שהוא מספר ולהגדיר לו שהוא nextInt, כלומר מחפש מספר. מה שיקרה עכשיו, ברגע שהרובוט יגיע לשורה הזו, הוא יפתח בפני המשתמש אפשרות להזין מספר. צריך להקליד ואז המספר שהוזן נשמר לתוך המשתנה num.

ועכשיו ניתן לקחת את המשתנה בשם num ולבצע עליו מניפולציות שונות. למשל, המספר שהזין המשתמש כפול 10 זה מה שיוצג למשתמש כל פעם מחדש. מלבד nextInt ניתן גם להשתמש בnext ששומר String.

נסו בעצמיכם תרגיל - לקלוט שם, לקלוט מספר ולהציג את השם ואת המספר כפול 2.





בנוס - הגרלת מספר רנדומלי

לעיתים בודקי תוכנה (QA) לא ירצו להשתמש בScanner, אלא דווקא בפונקציה שמגרילה מספר רנדומלי ובכך הבודקים יכולים לודא האם מספרים אקראיים מקריסים את התוכנה או לא.

למעשה, נכתוב את השורה הבאה ולקבוע כי יוחזר לתוך משתנה ה `double` שלנו מספר בין 0 ל1.

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

שימו לב - כאשר נרצה מספר חד ספרתי למשל נכפול ב10, דו ספרתי למשל נכפול ב100... וכדומה...

```
int num1 = (int)(Math.random()*10);  
int num2 = (int)(Math.random()*100);  
int num3 = (int)(Math.random()*1000);
```

כעת, כאשר נרצה לבצע ריצה על הקוד שלנו - לא נהיה חייבים בעצמינו להזין בכל פעם מספרים. אלא, לרוץ מהמספר 0 ועד המספר 1000 למשל (ניתן גם שליליים כמובן) בצורת לולאה (ראה ערך: מדריך הלוגיקה). ניתן כאמור גם להישאר עם Scanner והאפשרות להעניק למשתמש שלנו לכתוב בעצמו כמובן, אין הדבר סותר...



אובייקטים

ניתן ליצור מחלקות (class) נפרדות שלא ידברו עם console אלא יהיו שייכות לתוכנה שאתם בונים. הכוונה היא - נוכל ליצור class חדש לחלוטין שלא יקבל עכשיו הוראות של פלט למשתמש או את פונקציית הmain. אלא, class שלנו יכיל משתנים בעצמו, ובתוך class הראשי נדבר עם אותם משתנים. הדבר ייצור לנו סדר וארגון. לא נרצה שבתוך הmain והפונקציה הראשית שלנו יהיה לנו המון משתנים והרבה קוד מיותר. ניתן להשתמש בעקרון הפונקציות (ראה ערך פונקציות בשיעורי הלוגיקה) בג'אווה גם בשימוש של מחלקות נפרדות. כל מחלקה תכיל את המשתנים שלה ורק במחלקה הראשית נשלוף כל פעם מהמחלקה הרלוונטית את המשתנה הרלוונטי לנו באותה העת. נסו בעצמיכם, במקום ליצור משתנים בתוך הmain כדי לדבר עם הScanner צרו מחלקה חדשה לגמרי בשם Value. בתוך Value צרו משתנה של מספר לצורך הדוגמא.





```
public class Value{  
    public int num = 0;  
}
```

```
import java.util.Scanner;
```

```
public class first{  
    public static void main(){  
        Scanner scan = new Scanner(System.in);  
        num = scan.nextInt();  
        System.out.print('המספר שלך הוא : ' + num);  
    }  
}
```



תכנות מונחה עצמים

לאחר שהבנו את רעיון האובייקטים, ניתן לשדרג אותו ולבנות ממש מחלקות ואובייקטים שימשו אותנו ליצירת מסדי נתונים בצורה מובנית וחלקה יותר. הכוונה היא - צרו class חדש. הפעם אל תצרו את פונקציית הmain אלא class ישמש אותנו לאובייקט מסוים. האובייקט יכיל מספר תכונות ונוצר לשחק איתו. בואו נדגים על חתול. צרו class בשם cat. לכל חתול יש תכונות בסיסיות כמו צבע, גיל ושם. לכן, class שלנו ייראה תחילה כך:

```
public class cat{  
    private String color = 'white';  
    private int age = 3;  
    private String name = 'whiskey';  
}
```

השלב הבא שנרצה לעשות הוא לקלוט בmain הראשי שלנו, מהclass שמדבר עם המשתמש, תכונות של חתול על פי יצירת המשתמש וליצור את החתול בצורה מסודרת באמצעות האובייקט. עלינו ליצור Constructors (בנאו), כלומר פונקציה שלמה שעוסקת רק בקליטה של התכונות על ידי המשתמש. למעשה, הבנאי תמיד ייראה בהישענות על המשתנים הקיימים בclass מראש. ראו בסרטון.





```
public Cat(String color , int age , String name){  
    this.color = color;  
    this.age = age;  
    this.name = name;  
}
```

2 הערות:

1. כעת, יש לנו פונקציה בשם Cat שהיא public, כלומר, ניתן מ class אחר (מה class הראשי במקרה שלנו) לדבר איתה. לכתוב בו: `Cat('blue',1,'bub');` ויווצר חתול באופן וירטואלי שיש לו צבע כחול, הוא בן שנה וקוראים לו בוב.

2. המונח this מסמן את הבנת ההשמה בשלב הבנאי. אנו רוצים שתתבצע השמה, בתוך הסוגריים של הפונקציה אנו מקבלים ערך מהמשתמש כביכול ומזינים אותו לתוך המשתנה של האובייקט. הטכניקה הזו עובדת. כמו בהשמה. ה `this` מסמן את `color` של האובייקט ולא את `color` מהפונקציה. הרי לשתי המשתנים קוראים `color`. (למען ויתור על `this` ולהסיר את הבלבול, ניתן לשנות את שמות המשתנים).

צפו בסרטון מעלה והבינו איך ניתן להציג עכשיו את החתול בצורה דינאמית ויפה למשתמש. אנו רוצים שהוא ממש יראה שהוא מוסיף תכונות. המשיכו גם לשיעור הבא העוסק בפולימורפיזם. המשך של האובייקטים.



פולימורפיזם

בג'אווה אנו יכולים לבצע מצב של הורשה.

רעיון ההורשה הוא מתן אפשרות להרחבה של תוכנית אחת בעזרת תוכנית אחרת היורשת ממנה את תכונותיה ומסוגלת להוסיף ולשנות אותן.

אם נביט על החתול שיצרנו בשיעור הקודם במדריך, נוכל להוסיף לו אבא שיהיה מעליו. האבא יהיה כביכול 'יונק'. ואותו 'יונק' יכיל תכונות שקיימות לכלל היונקים. החתול יוכל לרשת מהיונק את התכונות הרלוונטיות.

בנוסף, יכול להיות גם אבא ל'יונק', כמו למשל 'חיה' באופן כללי... הרעיון הוא שלכל אובייקט, לא משנה מי האבא ומי הבן, יש את התכונות שלו. במידה ואובייקט יורש מאובייקט אחר אוטומטית הוא מקבל את התכונות של האב.

על מנת לבצע הורשה נצטרך לרשום בclass שיש הורשה ולציין ממי אנו יורשים. בואו נביט על המחלקות הבאות:



```
public class Animal{
    private Boolean mammal = true;
}

public Animal(Boolean mammal){
    this.mammal = mammal;
}

public class Cat extends Animal{
    private String color = 'white';
}

public Cat(String color, Boolean mammal){
    super(mammal);
    this.color = color;
}
```

מכאן, נוכל להסיק כי:

1. השימוש במילה super קורה בתוך הבנאי של הבן. הוא זה שמייבא את כל התכונות מהאב.
2. אנו יכולים פשוט להזין את שם התכונה שם האבא בסוגריים של הבנאי בבן ביחד עם תכונות הבן.
3. ניתן לבצע גם פולימורפיזם. הכוונה היא שניתן ליצור אובייקט של האב ולהציג את הבן:

```
Animal a1 = new Cat('white',true);
```

ובכך אנו מייצרים **חיה מסוג חתול**.

הנושא דורש תרגול ואימון ארוך טווח. מזמין אתכם להמשיך לצפות במדריך המצולם. בהצלחה.





מחלקה אבסטרקטית

בהמשך לשיעור 6 בו ראינו הורשות והענקת תכונות בין אובייקטים, בשיעור 7 נלמד על מחלקת אב אשר לא ניתן לייצר אותה. היא מחלקה מופשטת. באמירה 'לא ניתן לייצר אותה' אני מתכוון לכך שלא יהיה ניתן לבצע דרכה פולימורפיזם או ליצור אובייקט שלה. לצורך העניין, נכון לרגע זה אני יכול בקלות לכתוב את הקוד הבא בקוד main שלי:

```
Animal a1 = new Animal(true);
```

(שוב, בהתבסס על שיעור 6 ועל המחלקה של החיה עם הבנאי שמכיל רק משתנה בוליאני).

לכאורה, אנו יכולים לייצר חיה. אבל אין דבר כזה חיה. למעשה זה משהו מופשט. מה כן יש? יש חתול, יש כלב, יש ג'ירף. אבל חיה? באופן כללי? לא ממש הגיוני שניתן אפשרות בממשק שלנו ליצר חיות כלליות, אלא רק חיות ספציפיות. ולכן - ג'אוה מעניקים לנו את האפשרות להוסיף את המילה

abstract

מיד אחרי שם ה class ובכך למנוע מהמפתח/משתמש ליצור את המחלקה הזו. שימו לב - עדיין יהיה אפשר לבצע הורשות, עדיין יהיה אפשר להוריש את התכונות ולבצע

extends

במחלקות הבן... אבל לא תינתן אפשרות ליצור אובייקט יחיד של ה class האבסטרקטי.



```
public abstract class Shape {  
    private int ribs = 0;  
    private String color = 'white';  
}
```

אם נכתוב מחלקה מופשטת - לא נוכל ליצור אובייקט שלה, אך ניתן להרחיב אותה בעזרת מחלקה רגילה וליצור אובייקט של המחלקה הרגילה הזאת. מחלקה מופשטת יכולה להכיל גם שיטות מופשטות, כלומר - שיטות לא ממומשות (לא כתובות); שיטות שרק הכרזנו עליהן באמצעות הכותרת אך לא כתבנו בהן קוד כלשהו), המיועדות להתממש על ידי מחלקה יורשת כלשהי.

```
public abstract class Shape {  
    private int ribs = 0;  
    private String color = 'white';  
    public abstract void myMethod();  
}
```





בניית ממשק - interface

בהמשך לשיעור 8 אלמד אתכם פה על ממשקים (Interfaces). אם נשים לב כאשר נפתח מחלקה חדשה בbluej נוכל ליצור גם interface ולא class. הנושא בהמשך ישיר לנושא המחלקה האבסטרקטית, לכן חשוב קודם להיות סגורים עליו. בראיונות עבודה מאוד אוהבים לשאול מה ההבדל בין מחלקה אבסטרקטית לבין interface. אעשה לכם סדר. ראשית, ניצור interface בשם actions ונוסיף אליו מספר פונקציות שיבצעו פעולות שאותן יממש החתול. כמו שתיית חלב למשל. לאחר מכן, ניקח אובייקט רגיל (קרי החתול) ונכריז כי מחלקת החתול מממשת את הממשק actions. מיד אראה לכם איך כותבים זאת. באותו הרגע אשר הכרזנו על כך - החתול יהיה חייב לממש את כל השיטות המוזכרות באותו הממשק. לממשקים עצמם מותר להכיל אך ורק הכרזות על שיטות, ומשתנים קבועים - ממשקים אינם מחלקות ולכן אין בנאי. בהבדל ממחלקה אבסטרקטית שמשמשת כ'אבא' לכל דבר. וכעת לקוד עצמו: אצור עבורכם את הממשק וגם החתול יהיה חייב לממש את כל הפונקציות שלו בדרך הבאה:

```
interface Actions{
    public void drinkmilk();
}

public class Cat extends Animal implements Actions{
    private String color = 'white';
}
public Cat(String color, Boolean mammal){
    super(mammal);
    this.color = color;
}
public void drinkmilk(){
    System.out.println('DrinkMilk Now!!!');
}
}
```