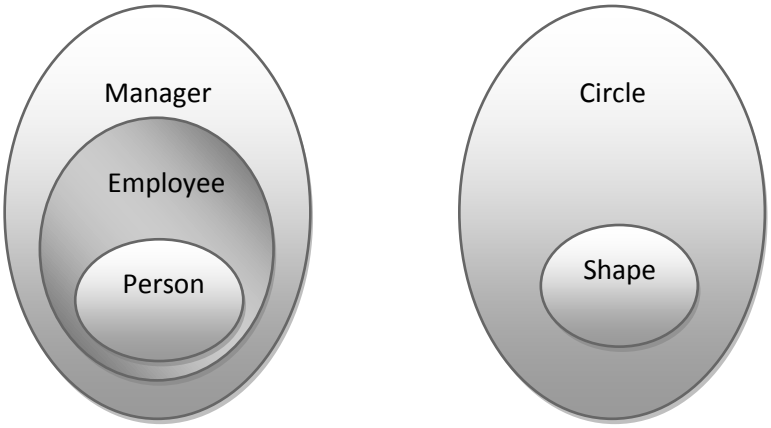
שעור 2. תכנות מונחה עצמים

הורשה

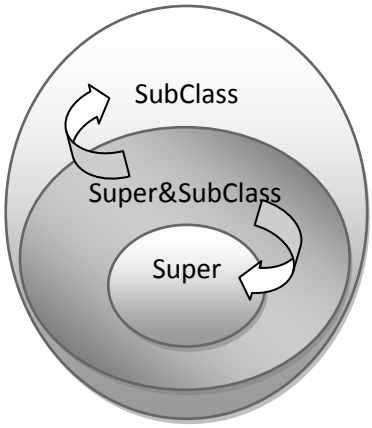
הורשה הינה דרך להגדרת מחלקות על ידי שימוש בתכונות ושיטות אשר הוגדרו על ידי מחלקות אחרות, כלליות יותר. מחלקה אשר יורשת ממחלקה אחרת, מקבלת ממנה את התכונות ואת השיטות שלה. פרט לתכונות ולשיטות שמתקבלות בהורשה, המחלקה עצמה יכולה להגדיר תכונות ושיטות נוספות. הורשה אינה קשר, היא מנגנון.



מנהל הוא סוג של עובד. עובד הוא סוג של אדם. לעובד יש תכונות של אדם ואת מרחב הפעולות שלו. בנוסף יש לו גם משכורת ומספר עובד. לעובד יש שיטות "כניסה למערכת" ועדכון נתונים במערכת, פעולות אשר אינן מוגדרות עבור אדם. מנהל הוא עובד. בנוסף לתכונות ופעולות העובד יש למנהל תכונות ופעולות נוספות.

הנקודה היא צורה. לצורה יש תכונה של צבע ופעולת צביעה. נקודה יורשת מצורה את תכונותיה, ומוסיפה עליהם שני שעורים X ו-Y. הפעולות הנוספות של נקודה הם: צייר, מחק, הזז, חשב מרחק עד ראשית הצירים.

המוריש	היורש
אבא	בן
Parent	Child
Father	Son
Base Class	Derived Class
Super Class	Subclass
מחלקת בסיס	מחלקה נגזרת



הגדרתה של מחלקה שיורשת ממחלקה אחרת נקרא גם בשם extending. האופן שבו מגדירים מחלקה שיורשת ממחלקה אחרת הוא כדלקמן:

בהמשך שורת הכותרת של המחלקה שמגדירים יש לרשום את המילה extends ומייד אחריה את שמה של המחלקה המורישה. רק לאחר מכן, בתוך הסוגרים המסולסלות של המחלקה המוגדרת ניתן להוסיף משתנים ומתודות אשר יתווספו לאלה שהגיעו בהורשה.

ב-JAVA (להבדיל מ- C++) כל מחלקה יכולה לרשת רק ממחלקה אחת. ב-JAVA אין הורשה מרובה (מצב שבו מחלקה נתונה יורשת ממספר מחלקות). תכנון התכנית תחת המגבלה שלא ניתן לרשת מיותר ממחלקה אחת יותר מבני ויותר מסודר. כאשר תכנון המחלקות נכון ומסודר, אין צורך בהורשה מרובה. עם זאת, אין כל מגבלה על מחלקה מלהוריש ליותר ממחלקה אחת. מחלקה יכולה להוריש למספר בלתי מוגבל של מחלקות.

כל מחלקה שמגדירים **ולא מציינים שהיא יורשת** ממחלקה אחרת תירש (כברירת מחדל) מהמחלקה Object את כל תכונותיה(המשתנים) ואת כל המתודות שמוגדרות בה. אם המחלקה שהגדרנו יורשת ממחלקה אחרת אז התכונות והמתודות שהוגדרו במחלקה Object יגיעו בהורשה מאותה מחלקה אחרת. אם גם המחלקה האחרת ירשה ממחלקה אז בדיקה של היררכית המחלקות תגלה מהר מאד כי במעלה ההיררכיה מגיעים לבסוף למחלקה שלא יורשת מאף אחת. מחלקה זו היא שתירש את התכונות ואת המתודות שהוגדרו במחלקה Object ותעבירן הלאה במורד ההיררכיה עד למחלקה שבוחנים. מסיבה זו, ניתן לומר, כי התכונות והמתודות שמוגדרות במחלקה Object קיימות, למעשה, בכל מחלקה.

המחלקה הנגזרת יכולה להגדיר מטודות חדשות או לתת משמעות חדשה למטודות שנורשו על ידי הגדרתם מחדש. בעולה זו מכונה **"דריסה" (Overriding)** . דוגמה: שיטה toString, המוגדרת במחלקת Object ונדרסת כמעט בכל מחלקת המשתמש. כאשר נקראת מתודה על עצם ממחלקה מסוימת, נבחרת הגרסה העדכנית ביותר של המטודה.

התכנית הבאה מתארת הורשה פשוטה בין שתי מחלקות: המחלקה מעגל יורשת מצורה: (נשתמש במילה השמורה extends בהגדרת המחלקה).

```
public class Shape{
            private String color;
            public Shape(){ this.color = "black";}
            public Shape( String color){this.color = color;}
            public String getColor( ) {return this.color;}
            public String toString( ) {return "color: "+color;}
            public double area(){return 0;}
            public double perimeter() {return 0;}
}
public class Circle extends Shape{
      public Point _center;
      public double _radius;

      public Circle(Point p, double radius, String color){
            super(color);
            _center = new Point(p.x(),p.y(), color);
            _radius = radius;
      }
      public Circle(Point p, double radius){
            _center = new Point(p);
            _radius = radius;
      }
      public Circle(String color){
            super(color);
            _center = new Point(0,0);
            _radius = 1;
      }
      public Circle(Circle c){
            super(c.getColor());
            _center = new Point(c._center);
            _radius = c._radius;
      }
      Point p(){return _center;}
      double r(){return _radius;}
      public double area( ){ return Math.PI*Math.pow(_radius, 2);}
      public double perimeter( ){return 2*Math.PI*_radius;}
      public String toString()  {
            return "center: "+_center.toString()+", radius: "+_radius+", "+super.toString();
      }

      public static void main(String[] args) {
            Point p = new Point(3,4);
            Point p1 = new Point(6,8);
```

```
        Circle cr = new Circle(p,5,"red");
        Circle cr0 = new Circle("blue");
        Circle cr1 = new Circle(p1,8);
        System.out.println(cr.toString());
        System.out.println(cr0.toString());
        System.out.println(cr1.toString());
    }
}
```

הרשאות הגישה.

ניתן להוסיף בתחילת שורת ההצהרה על משתנה/מתודה את אחת המלים הבאות: private, public, protected

בהוספת כל אחת מהמלים הנ"ל יש השפעה על אפשרות הגישה/השימוש במשתנה/מתודה.

הרשאת הגישה private

משתנה/מתודה שמוסיפים להם את הרשאת הגישה private בשורת ההצהרה שלהם – ניתן יהיה לגשת/להפעיל אליהם/אותם אך ורק בתוך מתודות ששייכות למחלקה שבה הוצהר על אותו משתנים/מתודה.

הרשאת הגישה public

משתנה/מתודה שמוסיפים להם את הרשאת הגישה public בשורת ההצהרה שלהם – ניתן יהיה לגשת/להפעיל אליהם/אותם בתוך כל המתודות הקיימות בכל המחלקות.

הרשאת הגישה protected

משתנה/מתודה שמוסיפים להם את הרשאת הגישה protected בשורת ההצהרה שלהם – ניתן יהיה לגשת/להפעיל אליהם/אותם אך ורק בתוך מתודות ששייכות למחלקה שבה הוצהר על אותם משתנים/מתודות או בתוך מחלקה ששייכת ל-package שאליו שייכת המחלקה שבה הוצהר על אותם משתנים/מתודות או בתוך מתודות ששייכות למחלקה שיורשת ממנה (באופן ישיר או עקיף).

אי מתן הרשאת גישה (package friendly)

משתנה/מתודה שלא מוסיפים להם הרשאת גישה בשורת ההצהרה שלהם – ניתן יהיה לגשת/להפעיל אליהם/אותם אך ורק בתוך מתודות ששייכות למחלקה שבה הוצהר על אותם משתנים/מתודות או בתוך מתודות ששייכות למחלקות אחרות ששייכות לאותו package שאליו שייכת המחלקה.

Access Levels				
Modifier	**Class**	**Package**	**Subclass**	**World**
public	Y	Y	Y	Y
protected	Y	Y	Y	N
no modifier	Y	Y	N	N
private	Y	N	N	N

המילה השמורה super

באמצעות מילה שמורה זו ניתן (במהלך פעולתה של מתודה שהוגדרה מחדש במחלקה overriding) להפעיל את המתודה בגרסתה במחלקה שמעל (במחלקה שהורישה למחלקה הנתונה). אופן השימוש במילה שמורה זו הוא כדלקמן: יש לרשום אותה, ומייד אחריה למקם נקודה מפרידה, ואחריה למקם קריאה להפעלת המתודה.

מתודות מסוג final

לשורת הכותרת של מתודה ניתן להוסיף את המילה השמורה final, ובכך למנוע את האפשרות שייעשה לה overriding. הודות להוספת מילה זו אל תחילת שורת הכותרת של המתודה (מייד אחרי הרשאת הגישה) לא ניתן יהיה להגדיר במחלקה היורשת את המתודה הזו מחדש. מתודה שהיא final מועברת בהורשה, מבלי שניתן יהיה להגדירה מחדש.

בהוספת המילה final לשורת הכותרת של מתודה יש שני יתרונות בולטים:

1. עלייה בביצועים. כאשר המתודה מוגדרת כמתודת final אין צורך לאתר את הגרסה המתאימה להפעלה כאשר יש קריאה להפעלתה (אין dynamically dispatching). למתודה שמוגדרת כמתודה מסוג final לא ניתן לעשות overriding, ולכן כשהמחשב נתקל בה הוא לא צריך לחפש את המתודה המסוימת שעליו להפעיל. בדרך זו מושג חיסכון בזמן.

2. הבטחה כי המתודה לא תשונה במחלקות שיורשות. כאשר מתודה מוגדרת כמתודת final מובטח שהיא לא תשונה במחלקות היורשות. לעתים, בהגדרתן של מחלקות זוהי תכונה נדרשת.

מחלקה מסוג final

את המילה final ניתן גם להוסיף לשורת הכותרת של מחלקה. המשמעות היא שלא ניתן יהיה להגדיר מחלקה אחרת אשר תירש ממנה. מחלקה שמוגדרת כמחלקה מסוג final תוכל לרשת אך לא להוריש.

את המילה final יש להוסיף לשורת הכותרת של המתודה מייד אחרי המילה שמייצגת את הרשאת הגישה שלה, ולפני המילה class. מחלקה מסוג final לא יכולה להיות abstract.

Polymorphism

פולימורפיזם– הוא יכולת של שיטה לעשות דברים שונים המבוססים על האובייקט היא פועלת על.

זהו העיקרון הבסיסי השלישי של תכנות מונחה עצמים. ***overloading***, ***overriding*** ו- ***dynamic method binding*** - שלושה סוגים של פולימורפיזם.

overloading שיטות הם שיטות עם שם זהה, אך מספר שונה של פרמטרים או סוגים שונים ברשימת הפרמטרים.

overriding שיטות עם אותה חתימה (שם זהה ופרמטרים זהים) המוגדרות מחדש ב-subclass.

dynamic method binding - יכולת של JVM לפתור הפניות לשיטות של subclass בזמן ריצת התכנית.

פעולתן של המתודות הבונות של מחלקות שיורשות מחלקות אחרות

Constructor Chaining

כאשר נוצר אובייקט ממחלקה שיורשת ממחלקה אחרת, בפעולת יצירתו של האובייקט החדש תהיה גם הפעלה של ה-constructor ששייך למחלקה המורישה בנוסף להפעלת ה-constructor ששייך למחלקה שממנה נוצר האובייקט.

אם המחלקה שממנה נוצר האובייקט יורשת ממחלקה אחרת ראשונה, והמחלקה האחרת הראשונה יורשת ממחלקה אחרת שנייה וכך הלאה. . . בפעולת יצירתו של האובייקט החדש יופעל כל אחד מה-constructors של כל אחת מהמחלקות.

הפעלת כל אחד מה-constructors נעשית באופן אוטומטי גם מבלי שב-constructors השונים תיכתב הקריאה להפעלתו של constructor מסוים. ה-constructor שיופעל כאשר אין קריאה ספציפית להפעלת constructor מסוים הוא ה- default constructor, ולכן, אם הוא לא קיים
(ה-default constructor , שהוא ה-constructor אשר לא מקבל ערכים בעת הפעלתו)
אז תתרחש שגיאת קומפילציה.

כדי לקרוא להפעלתו של constructor מסוים מבין ה-constructors שקיימים במחלקה המורישה יש לכתוב super ומייד אחריו בסוגריים את הערכים הנשלחים. על פי הערכים שנשלחים יופעל
ה-constructor המתאים.

התכנית הבאה מדגימה את אופן הקריאה להפעלתו של constructor מסוים באמצעות המילה super.

```
class A{
      A(){
            System.out.println("constructing A");
      }
}
class B extends A{
      B(){
            System.out.println("constructing B");
      }
}
 //כעת בכל פעם שנכתוב
```

```
B b = new B();
///נקבל על המסך את 2 השורות

constructing A
constructing B
```

ניתן באמצעות תכנית זו גם להבחין בכך שסדר הפעלת ה-constructors מתחיל במחלקה הבסיסית ביותר, המחלקה שמורישה לאחרות, המחלקה A, ואחר כך ממשיך לפי ההיררכיה במחלקות האחרות: B.

ירושה מנגנון הקריאה לשיטות

java יודעת (בזמן ריצה) כל אובייקט לאיזה מחלקה הוא שייך כאשר נפעיל שיטה של אובייקט:

- נחפש האם השיטה קיימת במחלקה (עליה שייך האובייקט)
- אם לא נמצא נחפש אצל המחלקה ממנה ירשנו (וכן הלאה).
- המנגנון הנ"ל נקרא polymorphism נדגים:

נעבור על מערך של צורות ונבקש להדפיס את השטח של כל צורה – השיטה שתופעל הינה של האובייקט האמיתי!!

ניהול טיפוסים בזמן אמת

למה צריך: נניח אנו מבקשים לדעת כמה עיגולים יש בתוך המערך של הצורות (שכולל גם משולשים ומלבנים).
נשתמש מילה השמורה instanceof, נדגים

```
if(arr[i] instinceof Circle)
```

instanceof הוא אופרטור בינארי שמקבל אובייקט (משמאל) ומחלקה (מימין) – מחזיר אמת אם האובייקט הינו מטיפוס של המחלקה. נשים לב התנאי הבא יחזיר תמיד אמת:

```
if(arr[i] instinceof Shape)
```

הקומפיילר אינו מריץ את הקוד ולכן אם רוצים להפעיל שיטה של מחלקה שאינה חלק מהמכנה המשותף יש צורך לבצע המרה, לטיפוס הספציפי.

דוגמא:

```
if(arr[i] instanceof Circle) {     // run time.
      Circle c= (Circle)arr[i];    // compile time.
      double rad = c.radius(); // return this radius
      double rad2 = arr[i].radius(); // Error (compile time)
      //....}
```

מחלקות אבסטרקטיות Abstract Classes

מחלקות אבסטרקטיות הן מחלקות מבניות (שלדיות) שלא ניתן לייצור מהן אובייקטים. הדרך המקובלת להגדרתה של המחלקה כמחלקה אבסטרקטית היא הוספת המילה abstract לשורת ההגדרה של המחלקה.

המחלקה האבסטרקטית משמשת כמחלקת בסיס, אשר ממנה מחלקות אחרות שנגדיר, יירשו. הגדרת המחלקה האבסטרקטית כוללת בתוכה הגדרות של משתנים ומתודות. המתודות יכולות להיות מתודות אבסטרקטיות, כלומר, מתודות שלדיות שבתחילת שורת הכותרת שלה מופיעה המילה abstract, ובסופה במקום בלוק מופיע ; . המתודה האבסטרקטית מתארת, למעשה, מתודה שחובה יהיה ליישם במחלקות היורשות.

המחלקה האבסטרקטית לא חייבת לכלול בתוכה רק מתודות אבסטרקטיות. היא בהחלט יכולה לכלול בתוכה גם מתודות שאינן אבסטרקטיות (מתודות עם גוף), והיא בהחלט יכולה לכלול בתוכה גם הגדרות של משתנים.

מהמחלקה האבסטרקטית לא ניתן לייצור אובייקטים. יש צורך להגדיר מחלקה אחרת אשר תירש מהמחלקה האבסטרקטית, ובה יש להגדיר באופן מסודר את המתודות שהוגדרו כמתודות אבסטרקטיות במחלקה האבסטרקטית.
כמובן מחלקה אבסטראקטית לא יכולה להיות מוגדרת כ- final.
המשמעות שיש למתודה אבסטרקטית דומה למשמעות שיש לפונקציה וירטואלית טהורה ב- C++.
<u>לדוגמא</u>. כדאי להגדיר מחלקת Shape כמפשטת, בגלל שיצירת אובייקט מסוג Shape–דבר שהוא חסר הגיון. שיטות לחישוב שטח והיקף מחזירות 0, אך זה לא מחייב לממש אותן במחלקה נגזרת. המנגנון של מחלקה מופשטת מחייב לממש את המתודות האלו.

מתודות סטטיות Static Methods

כאשר במחלקת הבן מוגדרת פונקציה סטטית עם אותה חתימה כמו פונקציה סטטית של מחלקת האב, אומרים שפונקציה של מחלקת הבן מסתירה (hiding) את פונקציה של מחלקת האב.

יש להבחין בין דריסת מטודה של מחלקת האב ע"י מתודה של מחלקת הבן לבין הסתרת פונקציה סטטית של מחלקת האב ע"י פונקציה סטטית של מחלקת הבן:

- כאשר דורסים מטודה של מחלקת האב, המתודה המופעלת היא מתודה של מחלקת הבן.
- בהסתרת פונקציה סטטית הפונקציה המופעלת תלויה אם היא מופעלת ממחלקת האב או ממחלקת הבן.

דוגמה:
המחלקה המורישה:

```
public class Animal {
    public static void testClassMethod() {
        System.out.println("The static method in Animal");
    }
    public void testInstanceMethod() {
        System.out.println("The instance method in Animal");
    }
}
```

המחלקה היורשת:

```
public class Cat extends Animal {
    public static void testClassMethod() {
        System.out.println("The static method in Cat");
    }
    public void testInstanceMethod() {
        System.out.println("The instance method in Cat");
    }

    public static void main(String[] args) {
        Cat myCat = new Cat();
        Animal myAnimal = myCat;
        Animal.testClassMethod();
        myAnimal.testInstanceMethod();
    }
}
```

הפלט:

```
The static method in Animal
The instance method in Cat
```

מהפלט רואים:

- כאשר משתמשים בדריסה הפונקציה המופעלת היא של מחלקת הבן (Cat).
- כאשר משתמשים בהסתרה הפונקציה המופעלת היא של מחלקת האב (Animal).