Inheritance
And Polymorphism I

OOP
תכונות עיקריות

- Encapsulation (ריכוזיות)
  - ✓ מאפשר הסתרת המבנה הפנימי של המחלקה
  - ✓ מאפשר בקרה נוחה על האובייקט (ממשק נוח)
- Inheritance (ירושה)
  - ✓ מאפשר שימוש חוזר בקוד (reuse)
  - ✓ מאפשר הרחבה של מבנה המחלקה מבלי לפגוע בתכונות הישנות שלה
- Polymorphism (רב צורתיות)
  - ✓ מאפשר התייחסות לעצמים שונים בתור דברים דומים

Inheritance
הורשה

- מנגנון (בזמן קומפילציה) שמאפשר להגדיר את המשותף שבין מספר מחלקות במחלקה אחת
- המחלקה הנגזרת יורשת את כל המשתנים והמתודות ממחלקת הבסיס
- ניתן להגדיר משתנים חדשים במחלקה הנגזרת
- ניתן להגדיר מתודות חדשות במחלקה הנגזרת
- ניתן לתת משמעות חדשה למתודות במחלקה הנגזרת וזאת ע"י הגדרתם מחדש במחלקה הנגזרת (מנגנון הדריסה – Override). במקרה כזה כאשר אנו פונים למתודה - נבחרת הגרסה העדכנית ביותר של המתודה
- המתודות של המחלקה הנגזרת קוראות למתודות של מחלקת הבסיס כדי שאלו תטפלנה באתחול משתנים ממחלקת הבסיס. כמו כן מתודות אלו מטפלות בעצמן במשתנים שהוגדרו במחלקה הנגזרת – דבר המאפשר את השימוש החוזר בקוד.
- גווה לא תומכת בירושה מרובה.

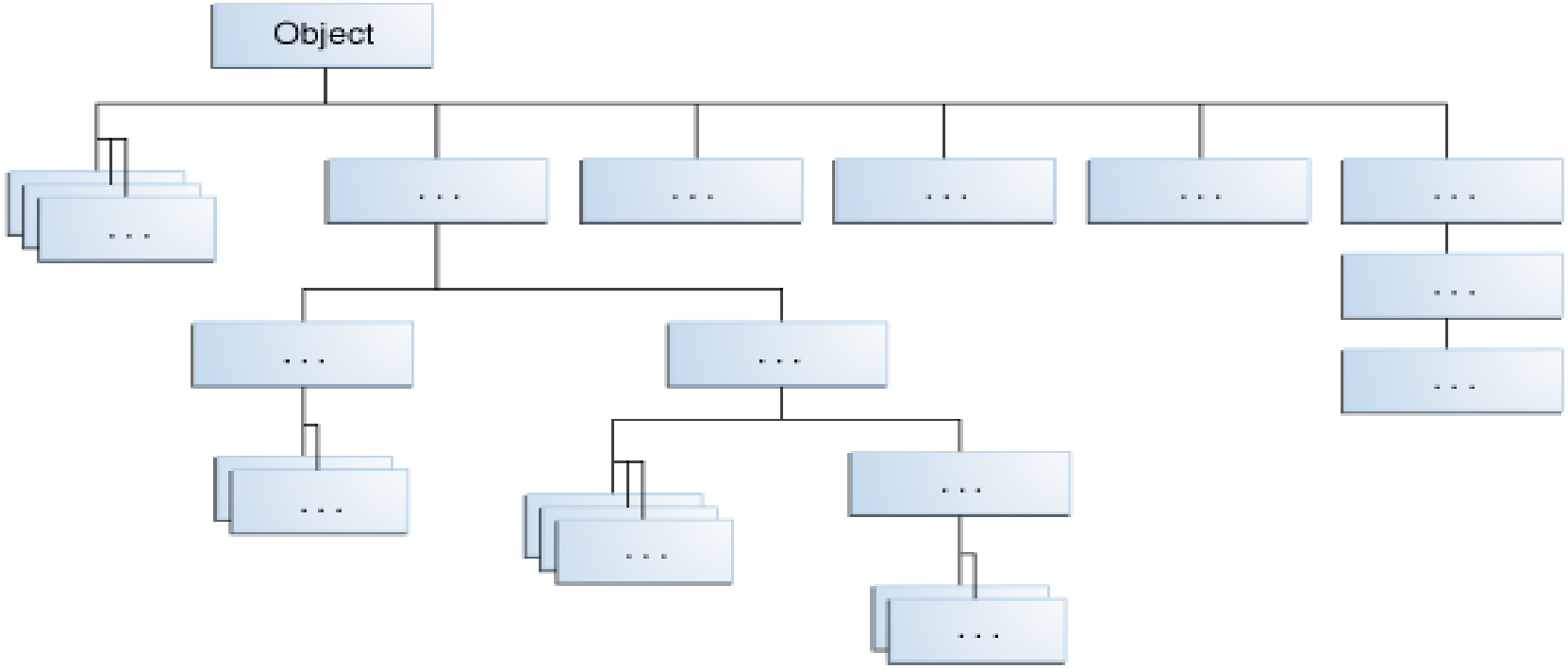
Inheritance
הורשה

- Inheritance is a technique of deriving new classes from existing classes.
- The methods and instance variables in the existing class can be reused or modified in the new class
- The existing claass is known as the BaseClass , SuperClass or ParentClass
- The new derived class from the BaseClass is known as the DerivedClass , SubClass, extended class or ChildClass.
- The Base-Derived class relationship defines a hierarchy structure, whereas each DerivedClass inherits the behavior and state of the BaseClass.
- Java does not support multiple inheritance.
- A new class (in java) can be derived only from one existing class.
- In general a DerivedClass has more functionality than its BaseClass
- A DerivedClass cannot access private members (instance variables and methods) of its BaseClass.
- A DerivedClass can access public and protected members of a BaseClass.
- A DerivedClass can access default members of a BaseClass - if in the same package.
- Inheritance is a compile-time mechanism

יתרונות
ההורשה

- דימוי של יחסים בין עצמים בעולם האמיתי
- מאפשר שימוש חוזר בקוד (reuse)
- מאפשר הרחבה של מבנה המחלקה מבלי לפגוע בתכונות הישנות שלה
- נוחות וקלות בביצוע שינויים
- חסכון בקוד(אין צורך לכתוב תכונות זהות לכל מחלקה ומחלקה)
- הגבלת גישה לנתונים (בטחון)
- מאפשר את קיומו של מנגנון הפולימורפיזם (ממשק אחיד למחלקות נגזרות)

The Java Platform
Class Hierarchy



- **The Object class, defined in the java.lang package, defines and implements behavior common to all classes - including the ones that you write. In the Java platform, many classes derive directly from Object, other classes derive from some of those classes, and so on, forming a hierarchy of classes.**
- **At the top of the hierarchy, Object is the most general of all classes. Classes near the bottom of the hierarchy provide more specialized behavior (Taken from: http://docs.oracle.com).**

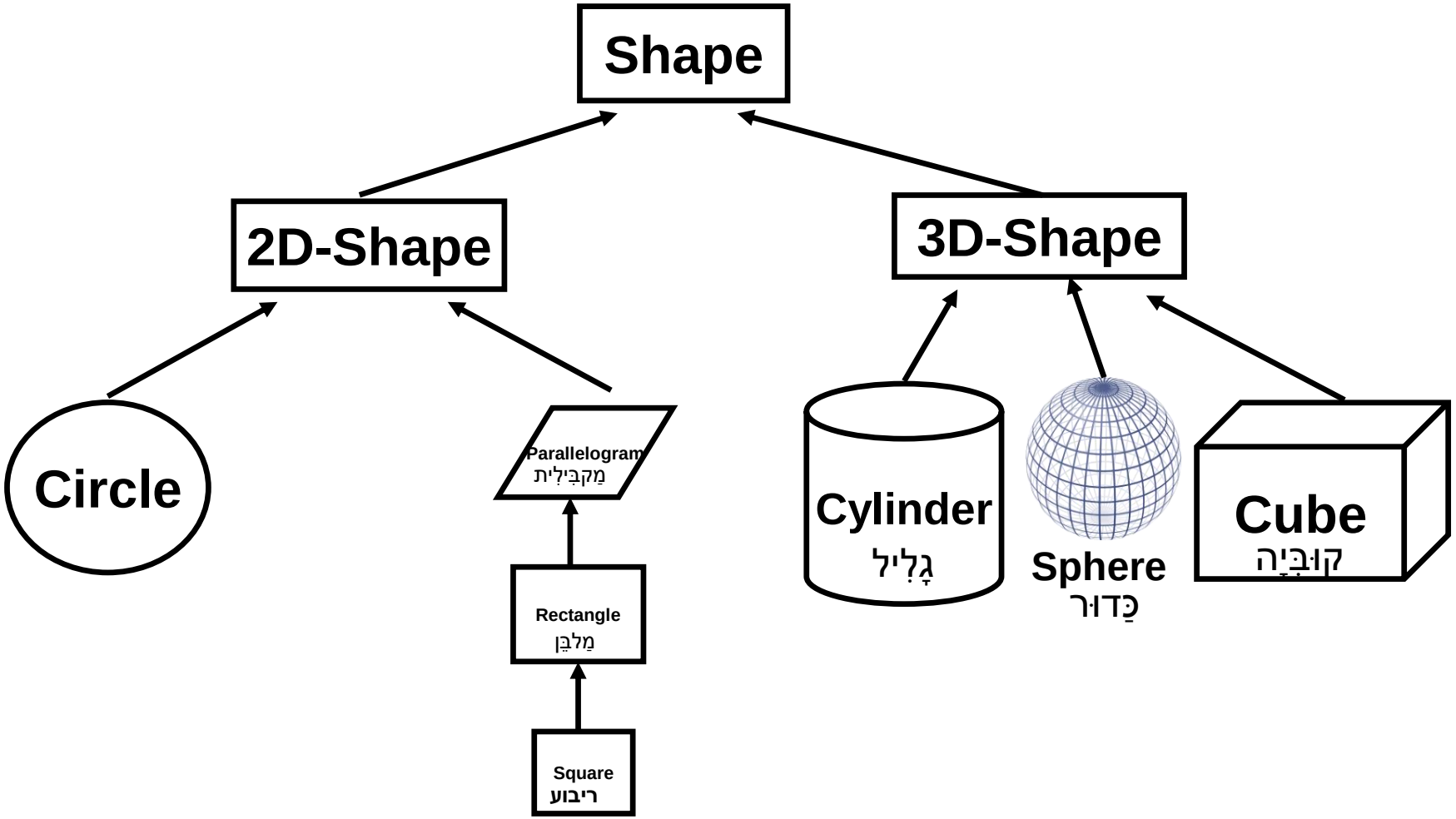
The Java Platform
Class Hierarchy

R
E
M
E
M
B
E
R

- Every class (except Object, which has no superclass) has one and only one direct superclass (single inheritance). In the absence of any other explicit superclass, every class is implicitly a subclass of Object.

- Classes can be derived from classes that are derived from classes which are derived from classes, and so on, and ultimately derived from the topmost class, Object. Such a class is said to be *descended* from all the classes in the inheritance chain stretching back to Object.

http://docs.oracle.com/javase/tutorial/java/IandI/subclasses.html


Shape
2D-Shape
3D-Shape
Circle
Parallelogram
מַקבִּילִית
Rectangle
מַלבֵּן
Square
ריבוע
Cylinder
גֶּלִיל
Sphere
כַּדוּר
Cube
קוּבִּיָה

Using The Keyword **extends** To Inherit a New Class

```
public class derived-class-name extends base-class-name {
        all members (instance variables and methods) of the Base-class
                                  +
                  new members of the Derived-class
}
```

derived class methods extend and possibly override those of the base class

The BasicRobot Class

Attributes

xLocation	direction
yLocation	numBeepers

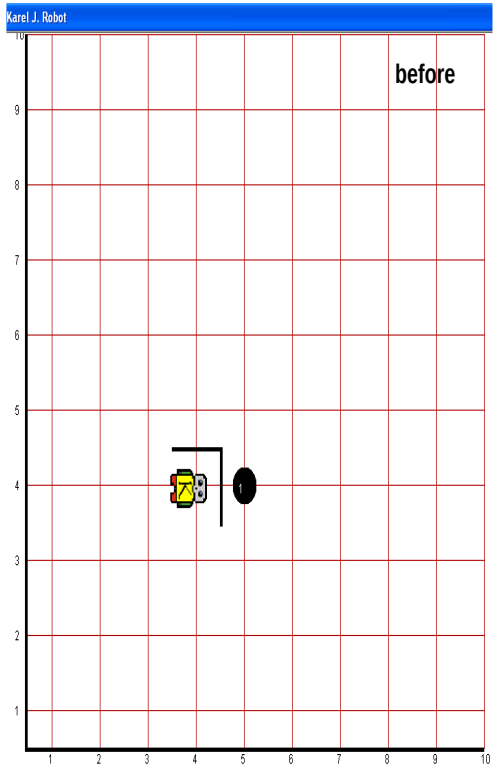
Methods

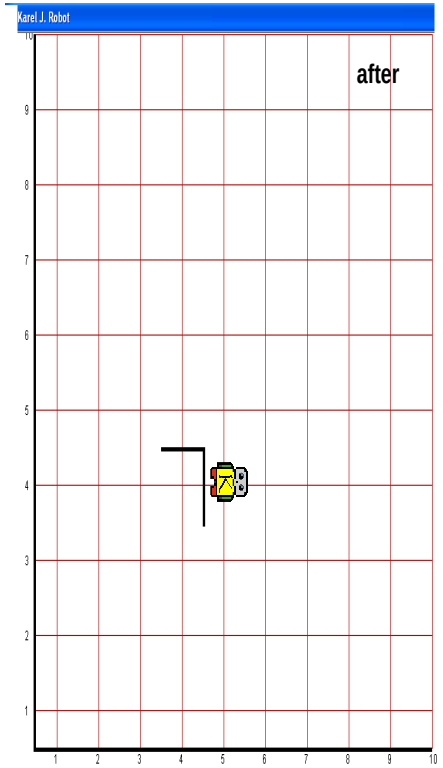
void move()
void turnLeft()
void pickBeeper()
void putBeeper()
void turnOff()
boolean facingNorth()
boolean facingSouth()
boolean facingEast()
boolean facingWest()
boolean anyBeepersInBag()
int xLoc()
int yLoc()
Direction facing()
BasicRobot (xLoc, yLoc, direction, beepersNum)

constructor

דוגמה

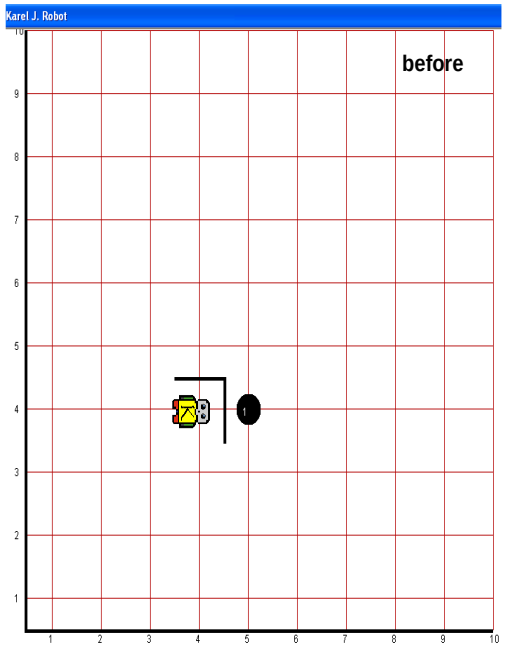
```
public static void main ( String [ ] args )  {
   BasicRobot rob = new BasicRobot ( 4 , 4 , East , 5 );
   rob.turnLeft();
   rob.turnLeft();
   rob.move();
   rob.turnLeft();
   rob.move();
   rob.turnLeft();
   rob.move();
   rob.move();
   rob.turnLeft();
   rob.move();
   rob.pickBeeper();
   rob.turnLeft();
   rob.turnLeft();
   rob.turnLeft();
   rob.turnOff();
}
```

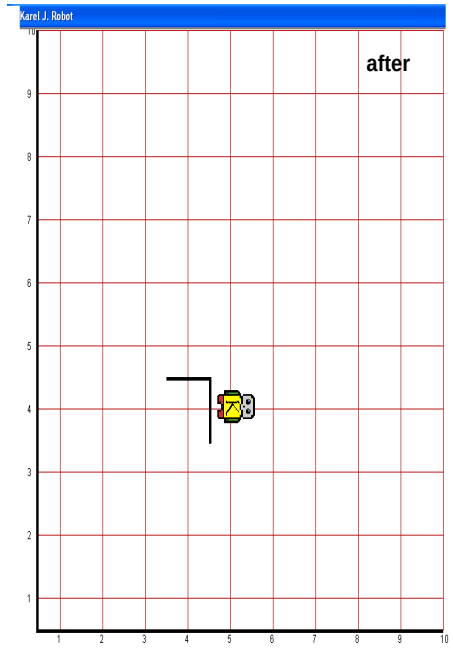




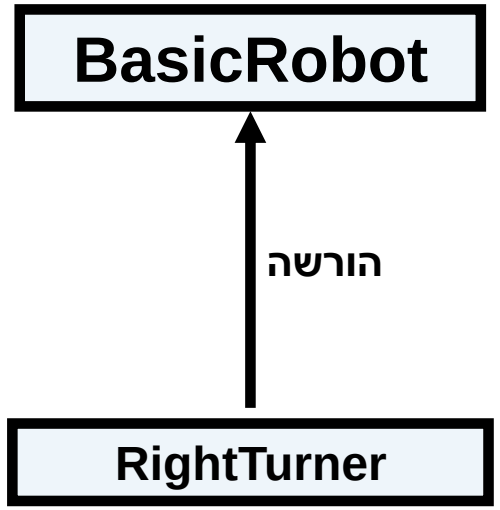
בהנחה שהרובוט
יודע להסתובב ימינה
איך הפתרון יראה?

```
public static void main ( String [ ] args ) {
  RightTurner rob = new RightTurner ( 4 , 4 , East , 5 );
  rob.turnRight();
  rob.move();
  rob.turnLeft();
  rob.move();
  rob.turnLeft();
  rob.move();
  rob.pickBeeper();
  rob.turnRight();
  rob.turnOff();
}
```





איך ניתן ליצור רובוט כזה?




```
class RightTurner extends BasicRobot {
    public RightTurner (int x, int y, Direction d, int b) {
        super (x, y, d, b);
    }
    public void turnRight() {
        turnLeft();
        turnLeft();
        turnLeft();
    }
}
```

משתמשים בהורשה

שימו לב :

אם מחלקה א' יורשת (נגזרת) ממחלקה ב' אז מחלקה א' מקבלת את כל התכונות והשיטות של מחלקה ב'.ההפך לא נכון

```
public class Person {
   protected String name;
   public Person(String name) {
      this.name=name;
      System.out.println("Person constructor has been generated");
   }
   public void whoAmI(){
      System.out.println("My name is " + name);
   }
   public String toString(){
      return name;
   }
}
```

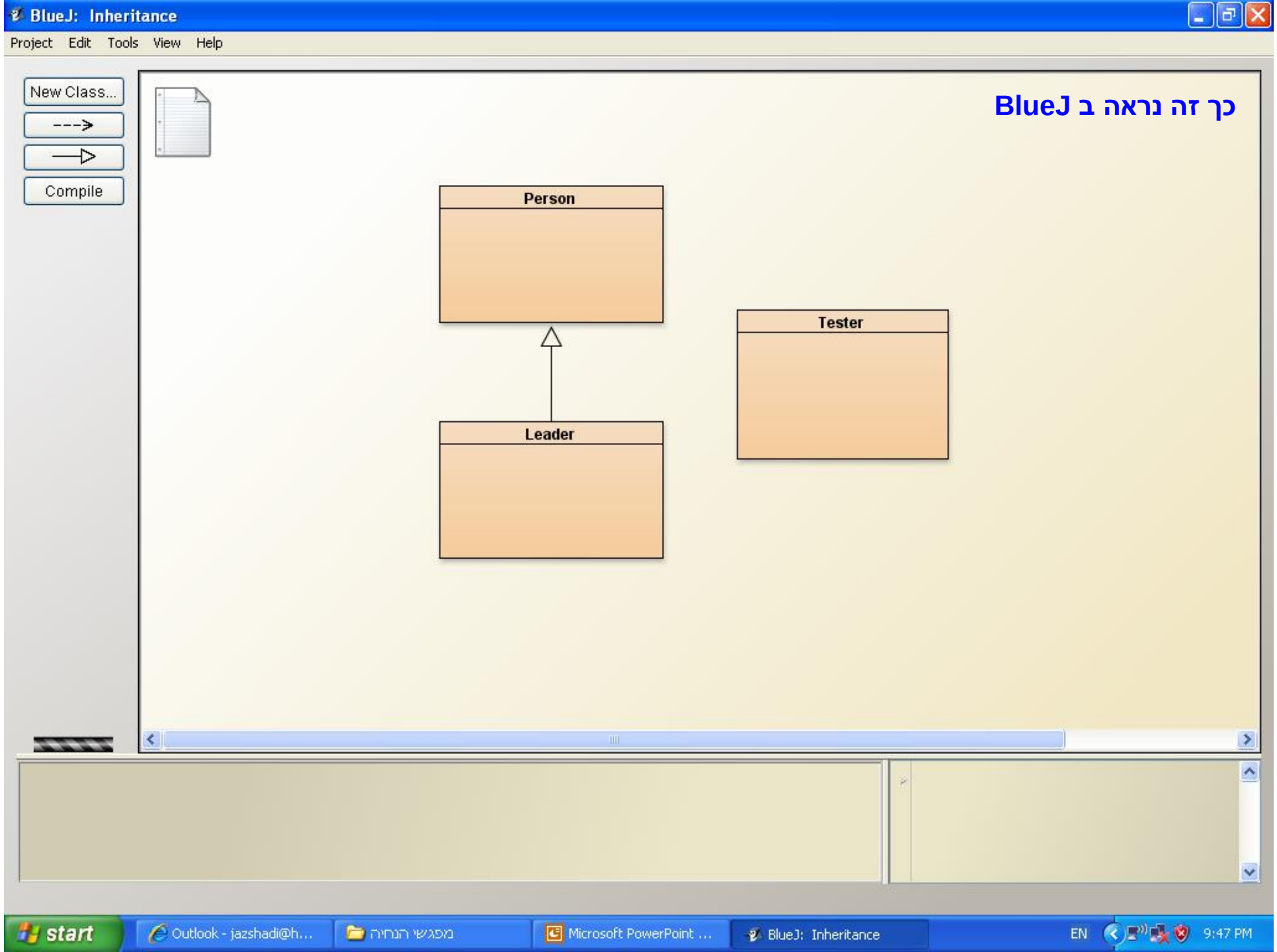
```
public class Leader extends Person {
   private int seniority;
   public Leader(String name,int seniority) {
      super(name);
      this.seniority=seniority;
      System.out.println("Leader constructor has been generated");
   }
   public String toString() {
     return "Leader " + name + " with seniority " + seniority;
   }
}
```

תרגיל

נתונה המחלקה Person והמחלקה Leader אשר יורשת ממנה. כמו כן נתון Tester בעמוד הבא. יש לבצע מעקב תוך כדי פירוט המהלכים שמתרחשים בכל שלב ושלב .

```
public class Tester {
    public static void main(String[ ] args){
        Leader L1=new Leader("Peres", 40);
        L1.whoAmI();
        System.out.println(L1);
    }
}
```

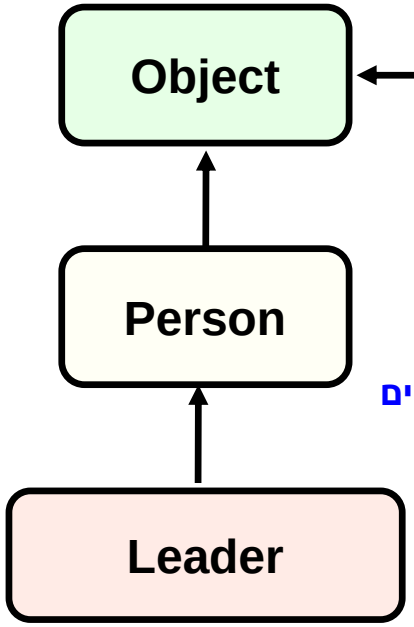
```
Output:
Person constructor has been generated
Leader constructor has been generated
My name is Peres
Leader Peres with seniority 40
```



כך זה נראה ב BlueJ

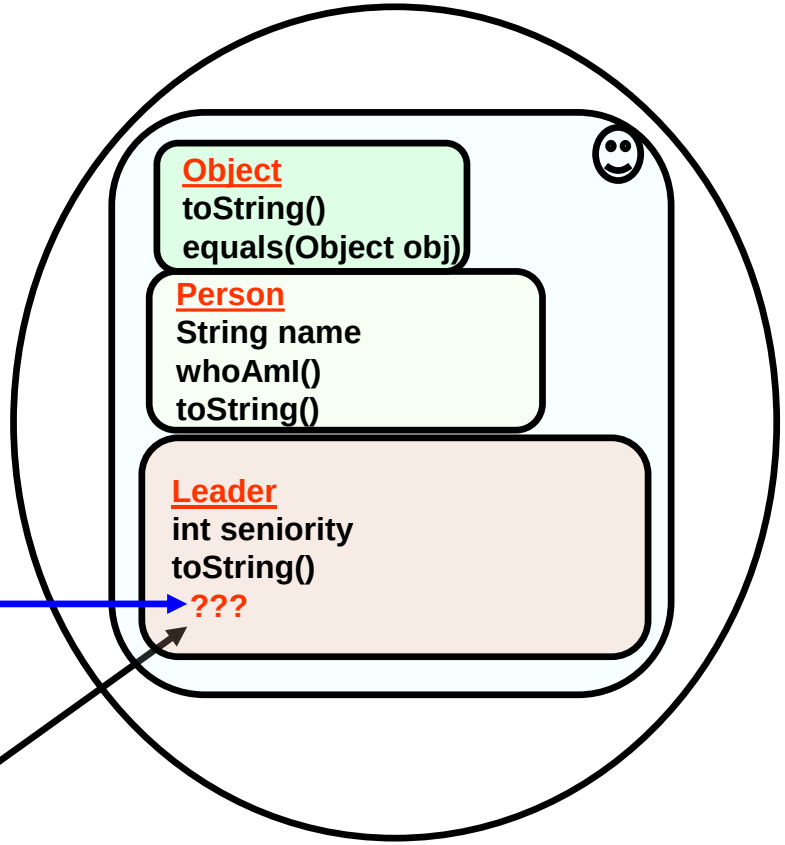
אובייקט מסוג Leader מכיל
בתוכו גם חלק של Person
וגם חלק של Object

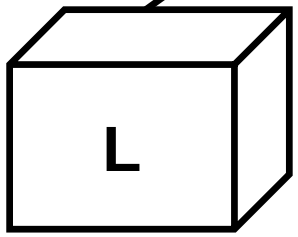
Leader L = new Leader(…)



כל מחלקה יורשת
במישרין או בעקיפין
מהמחלקה Object

בנוסף לשיטות שהוגדרו ב Leader
המחלקה Leader יורשת את כל השטות
שב Object ו Person וגם את כל המשתנים

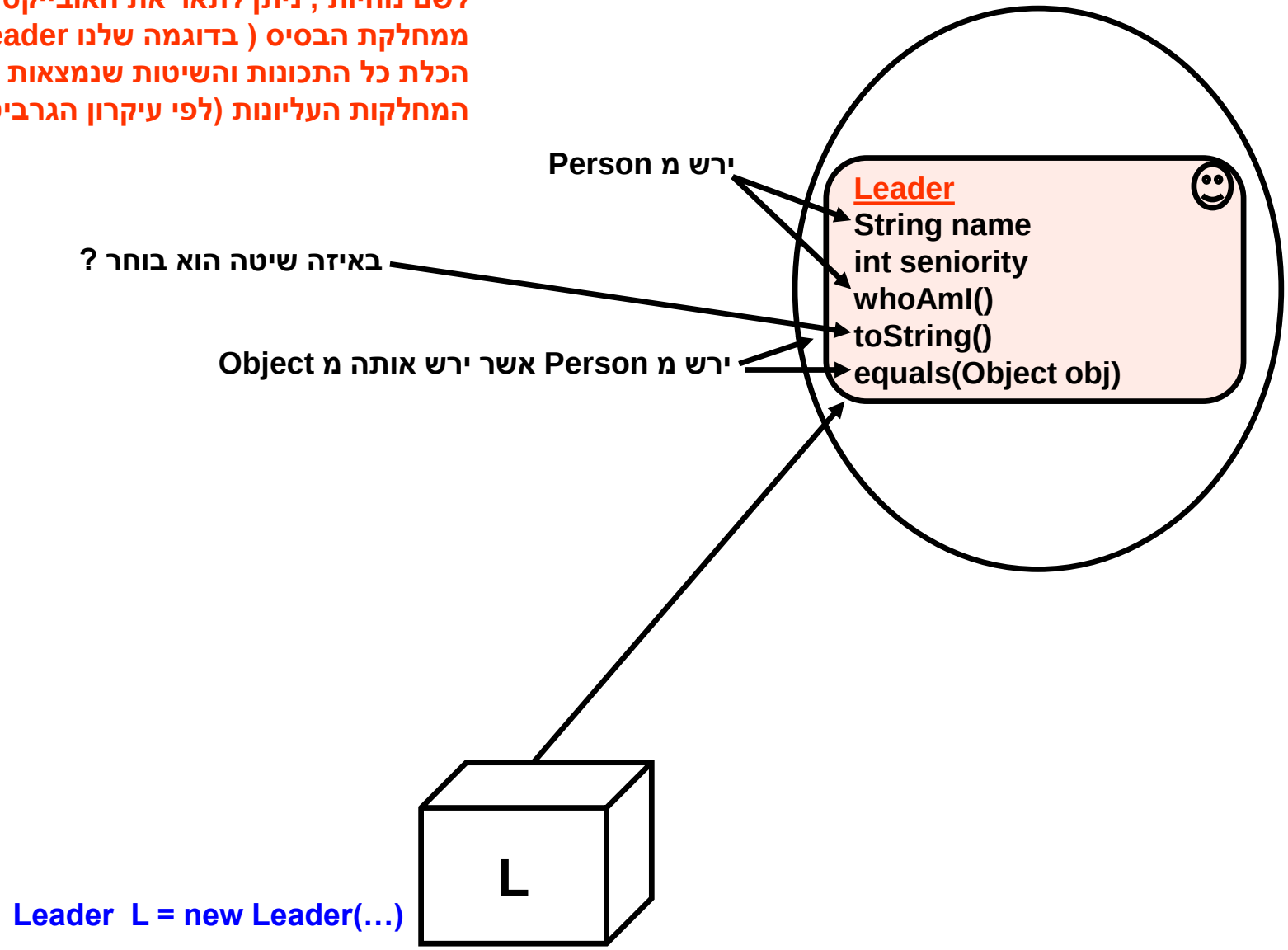




Leader L = new Leader(…)

נוצר אובייקט אשר מורכב משלושת
האובייקטים ביחד והמצביע מצביע על
החלק של Leader
שאלה :האם ניתן לכוון את המצביע על
חלק אחר באובייקט החדש?
תשובה: כן (פולימורפיזם)

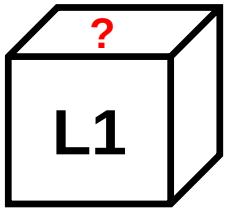
לשם נוחיות , ניתן לתאר את האובייקט שנגזר
ממחלקת הבסיס (בדוגמה שלנו Leader) ע"י
הכלת כל התכונות והשיטות שנמצאות בכל
המחלקות העליונות (לפי עיקרון הגרביטציה)



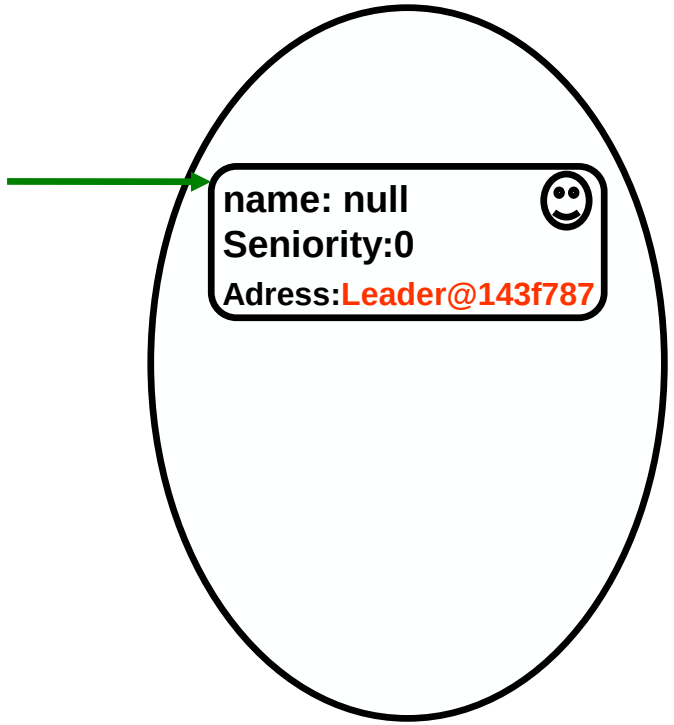
```
public class Tester {
    public static void main(String[] args){
        Leader L1;
        L1=new Leader("Peres", 40);
        L1.whoAmI();
        System.out.println(L1);
    }
}
```

מעקב

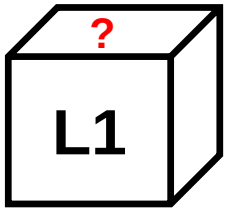
```
public class Tester {
    public static void main(String[] args){
        Leader L1;
        L1=new Leader("Peres", 40);
        L1.whoAmI();
        System.out.println(L1);
    }
}
```



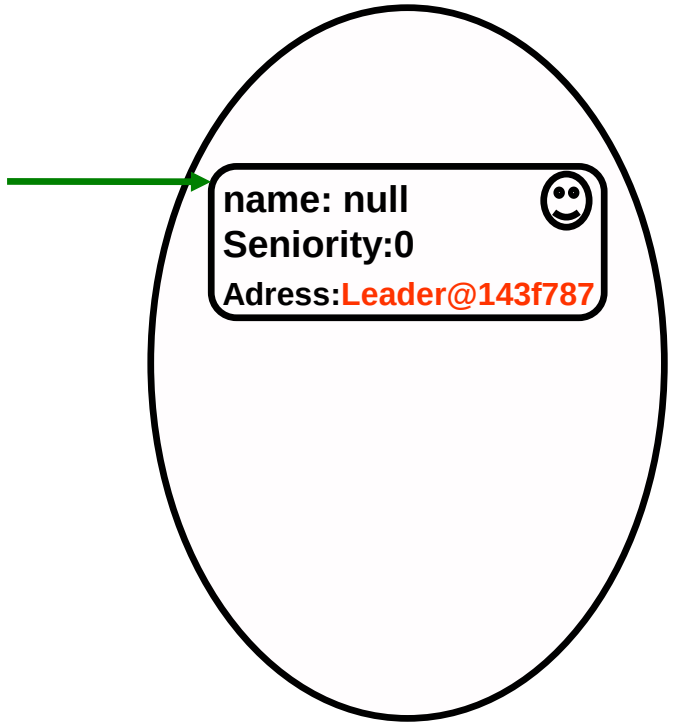
```
public class Tester {
    public static void main(String[ ] args){
        Leader L1;
        L1=new Leader("Peres", 40);
        L1.whoAmI();
        System.out.println(L1);
    }
}
```



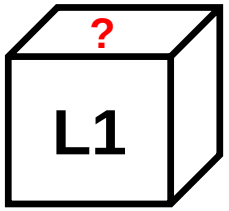

```
public Leader(String name, int seniority){
        super(name);
        this.seniority=seniority;
        System.out.println("Leader constructor has been generated");
    }
```



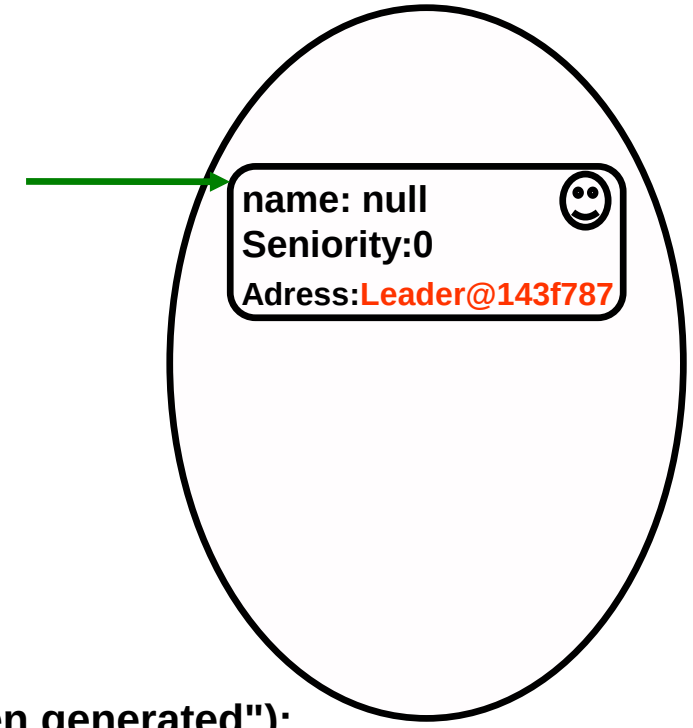
```
public class Tester {
    public static void main(String[] args){
        Leader L1;
        L1=new Leader("Peres", 40);
        L1.whoAmI();
        System.out.println(L1);
    }
}
```



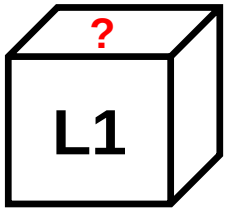

```
public Leader(String name,int seniority){
        super(name);
        this.seniority=seniority;
        System.out.println("Leader constructor has been generated");
}
```



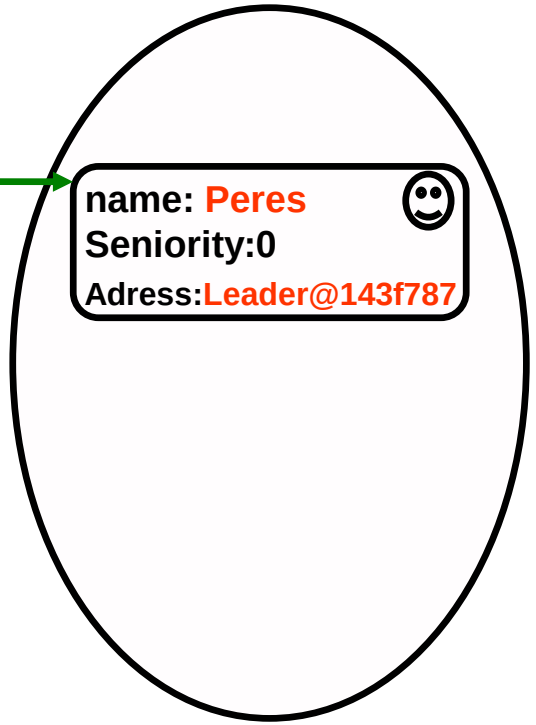
```
public class Tester {
    public static void main(String[] args){
        Leader L1;
        L1=new Leader("Peres", 40);
        L1.whoAmI();
        System.out.println(L1);
    }
}
```



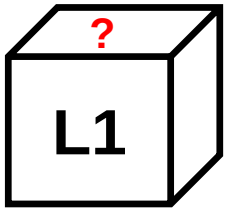

```
public Leader(String name,int seniority){
        super(name);
        this.seniority=seniority;
        System.out.println("Leader constructor has been generated");
    }

public Person(String name){
        this.name=name;
        System.out.println("Person construcor has been generated!");
    }
```



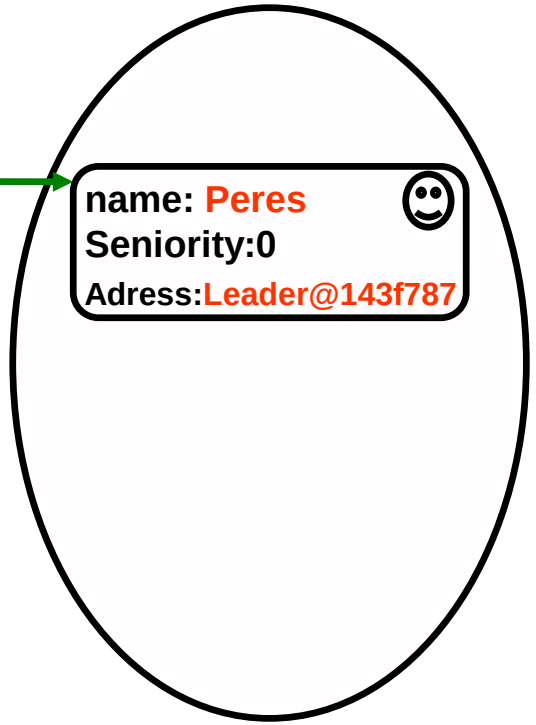
```
public class Tester {
    public static void main(String[] args){
        Leader L1;
        L1=new Leader("Peres", 40);
        L1.whoAml();
        System.out.println(L1);
    }
}
```



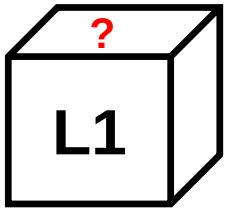

```
public Leader(String name,int seniority){
        super(name);
        this.seniority=seniority;
        System.out.println("Leader constructor has been generated");
}
public Person(String name){
        this.name=name;
        System.out.println("Person construcor has been generated!");
}
```



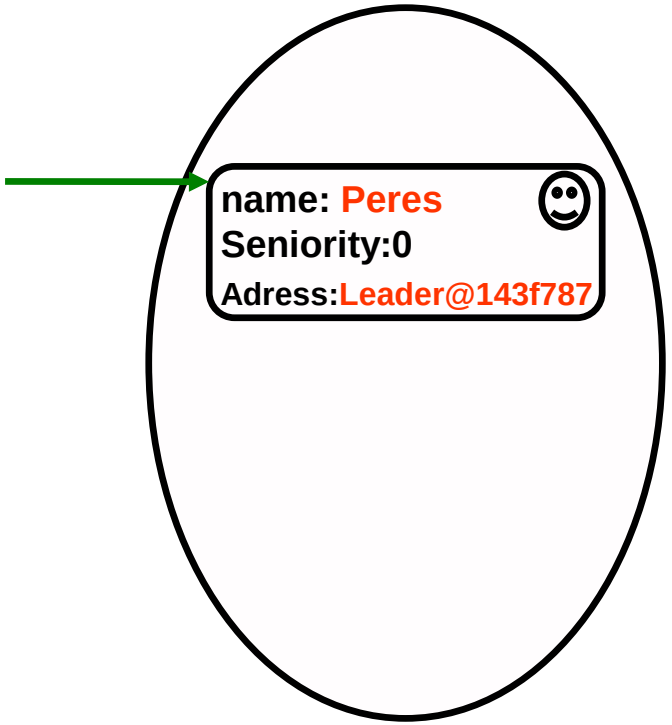
```
public class Tester {
    public static void main(String[] args){
        Leader L1;
        L1=new Leader("Peres", 40);
        L1.whoAml();
        System.out.println(L1);
    }
}
```



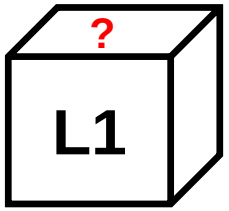

```
public Leader(String name,int seniority){
        super(name);
        this.seniority=seniority;
        System.out.println("Leader constructor has been generated");
}
public Person(String name){
        this.name=name;
        System.out.println("Person construcor has been generated!");
    }
```



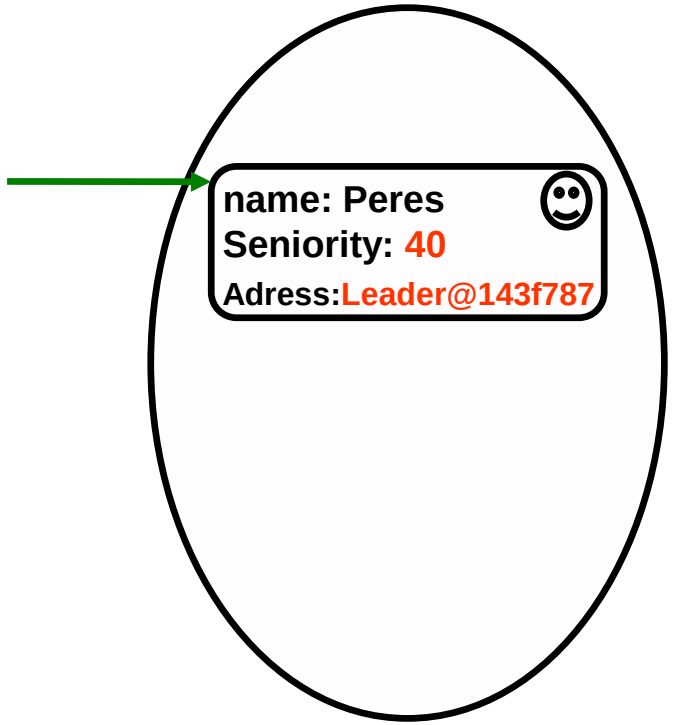
```
public class Tester {
    public static void main(String[] args){
        Leader L1;
        L1=new Leader("Peres", 40);
        L1.whoAml();
        System.out.println(L1);
    }
}
```



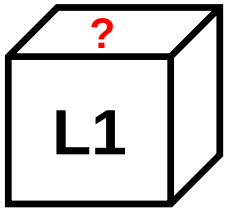

```
public Leader(String name,int seniority){
        super(name);
        this.seniority=seniority;
        System.out.println("Leader constructor has been generated");
}
public Person(String name){
        this.name=name;
        System.out.println("Person construcor has been generated!");
    }
```



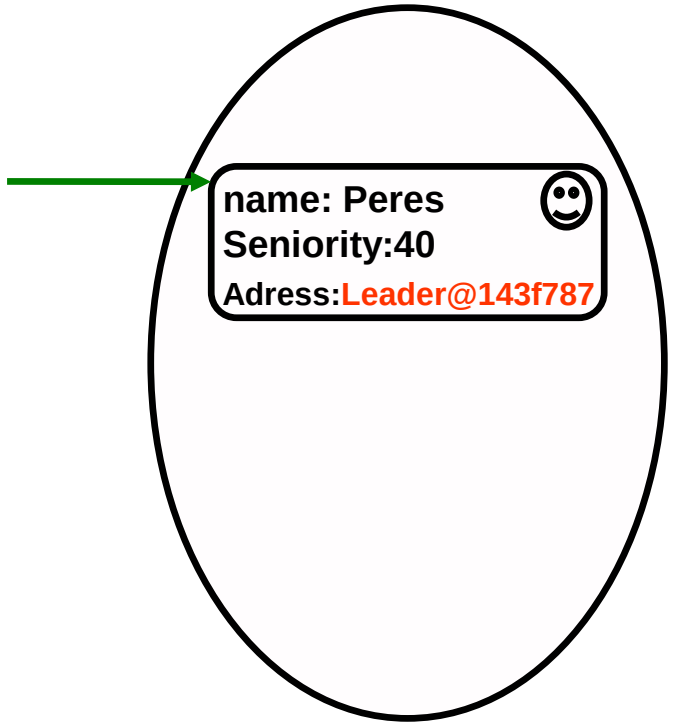
```
public class Tester {
    public static void main(String[] args){
        Leader L1;
        L1=new Leader("Peres", 40);
        L1.whoAml();
        System.out.println(L1);
    }
}
```



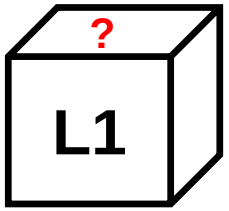

```
public Leader(String name,int seniority){
        super(name);
        this.seniority=seniority;
        System.out.println("Leader constructor has been generated");
}
```



```
public class Tester {
    public static void main(String[ ] args) {
        Leader L1;
        L1=new Leader("Peres", 40);
        L1.whoAml();
        System.out.println(L1);
    }
}
```

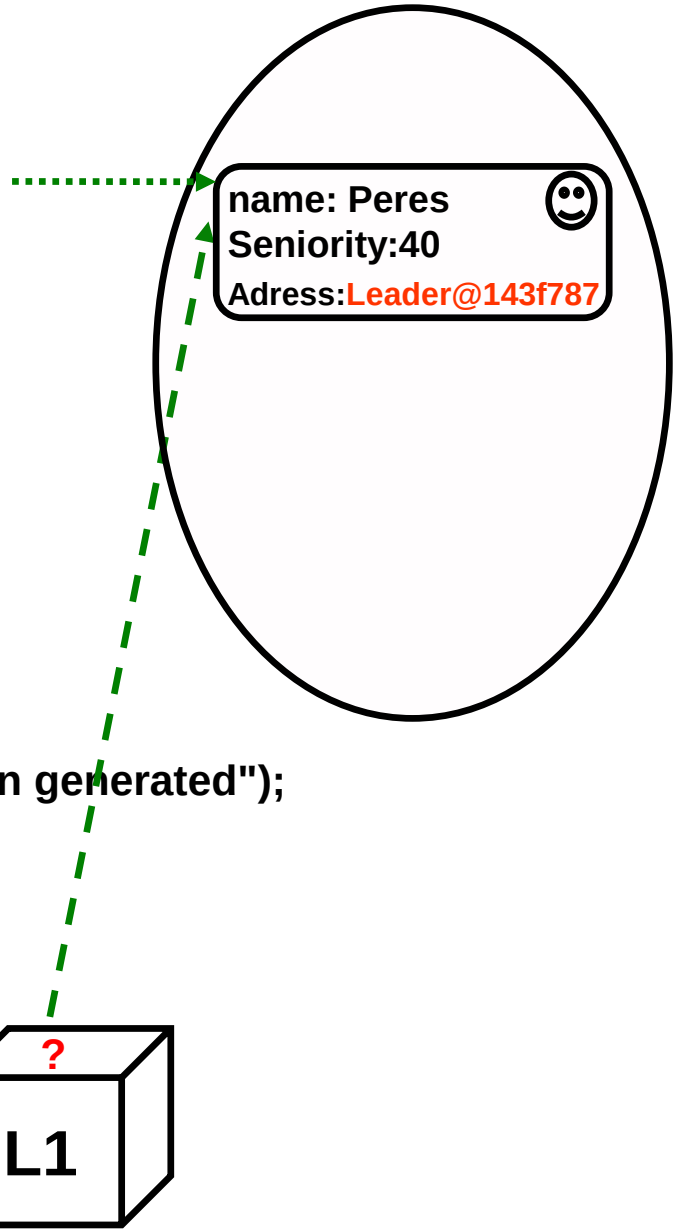



```
public Leader(String name,int seniority){
        super(name);
        this.seniority=seniority;
        System.out.println("Leader constructor has been generated");
}
```

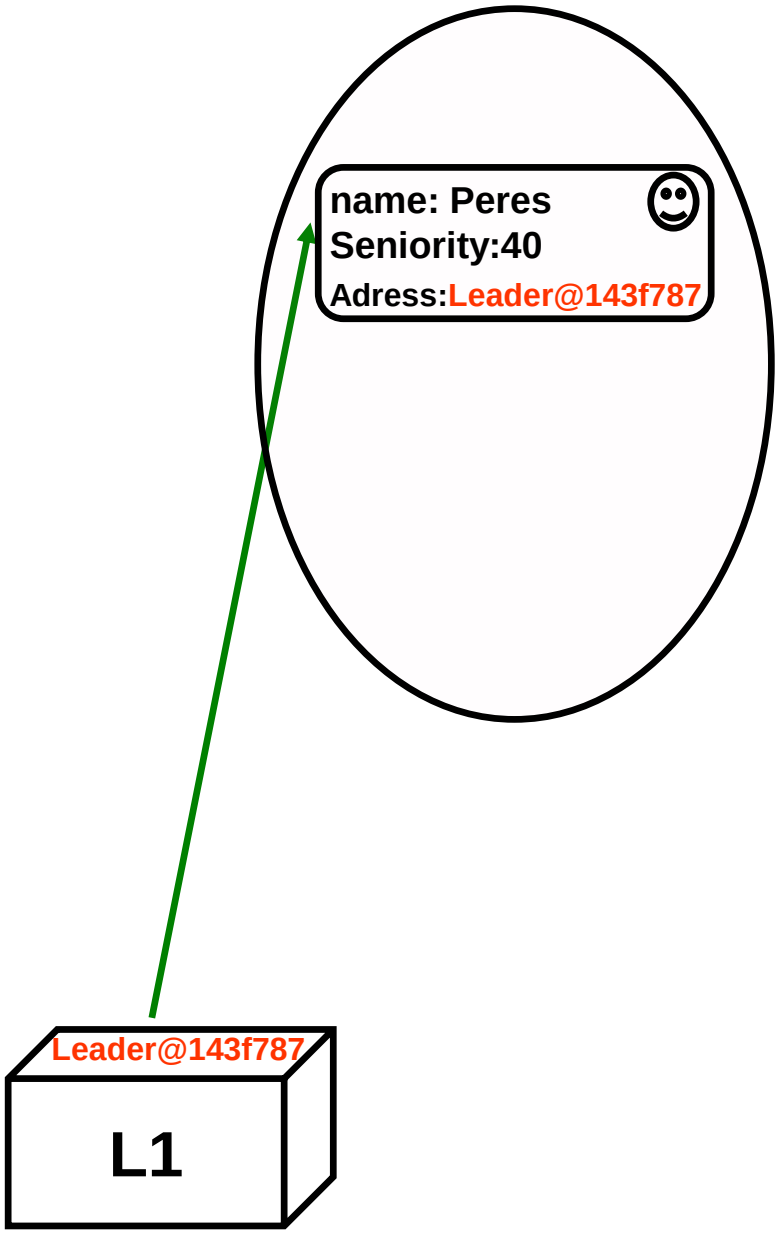


```
public class Tester {
    public static void main(String[ ] args){
        Leader L1;
        L1=new Leader("Peres", 40);
        L1.whoAml();
        System.out.println(L1);
    }
}
```

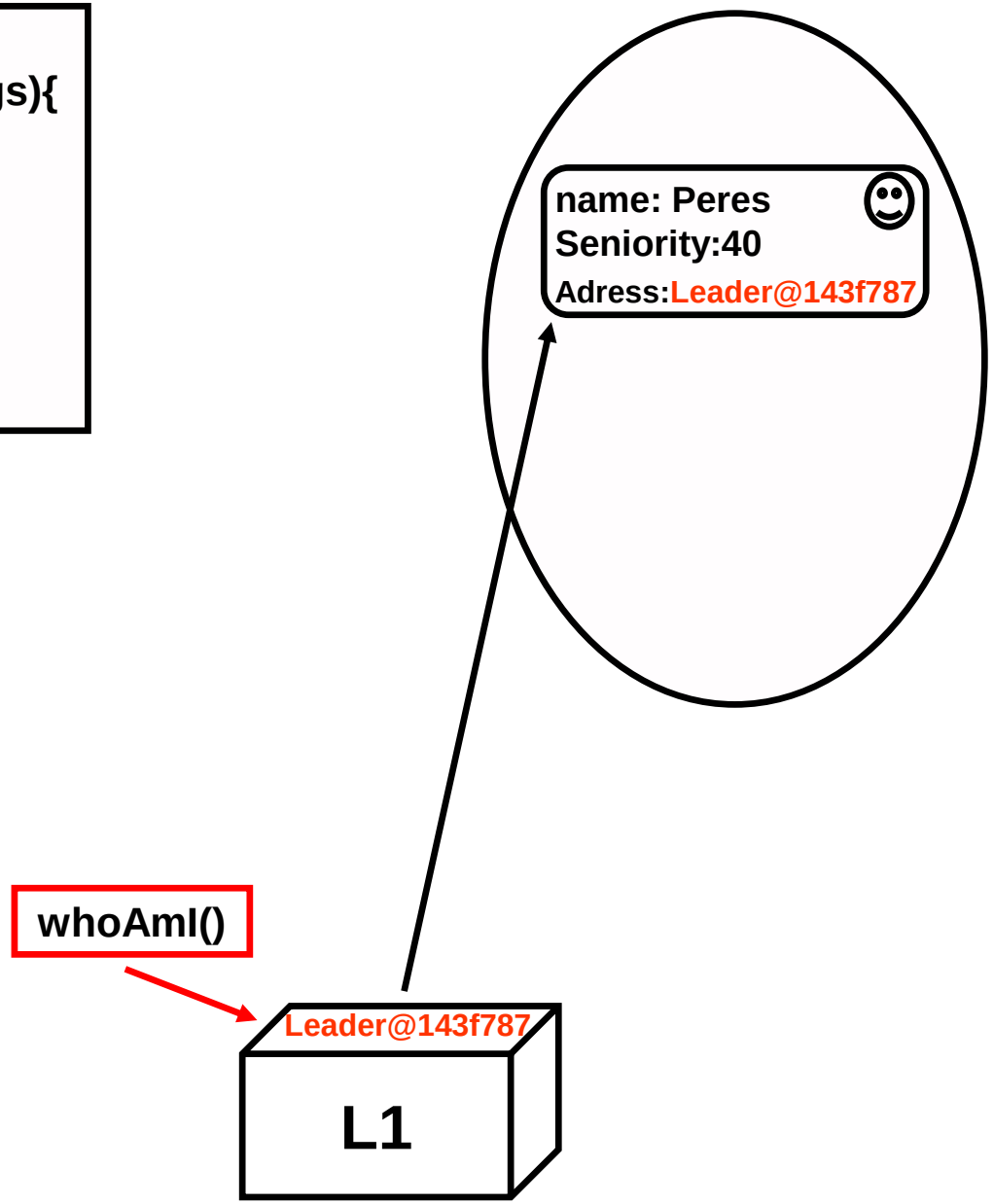
```
public Leader(String name,int seniority){
        super(name);
        this.seniority=seniority;
        System.out.println("Leader constructor has been generated");
}
```



```
public class Tester {
    public static void main(String[ ] args){
        Leader L1;
        L1=new Leader("Peres", 40);
        L1.whoAmI();
        System.out.println(L1);
    }
}
```

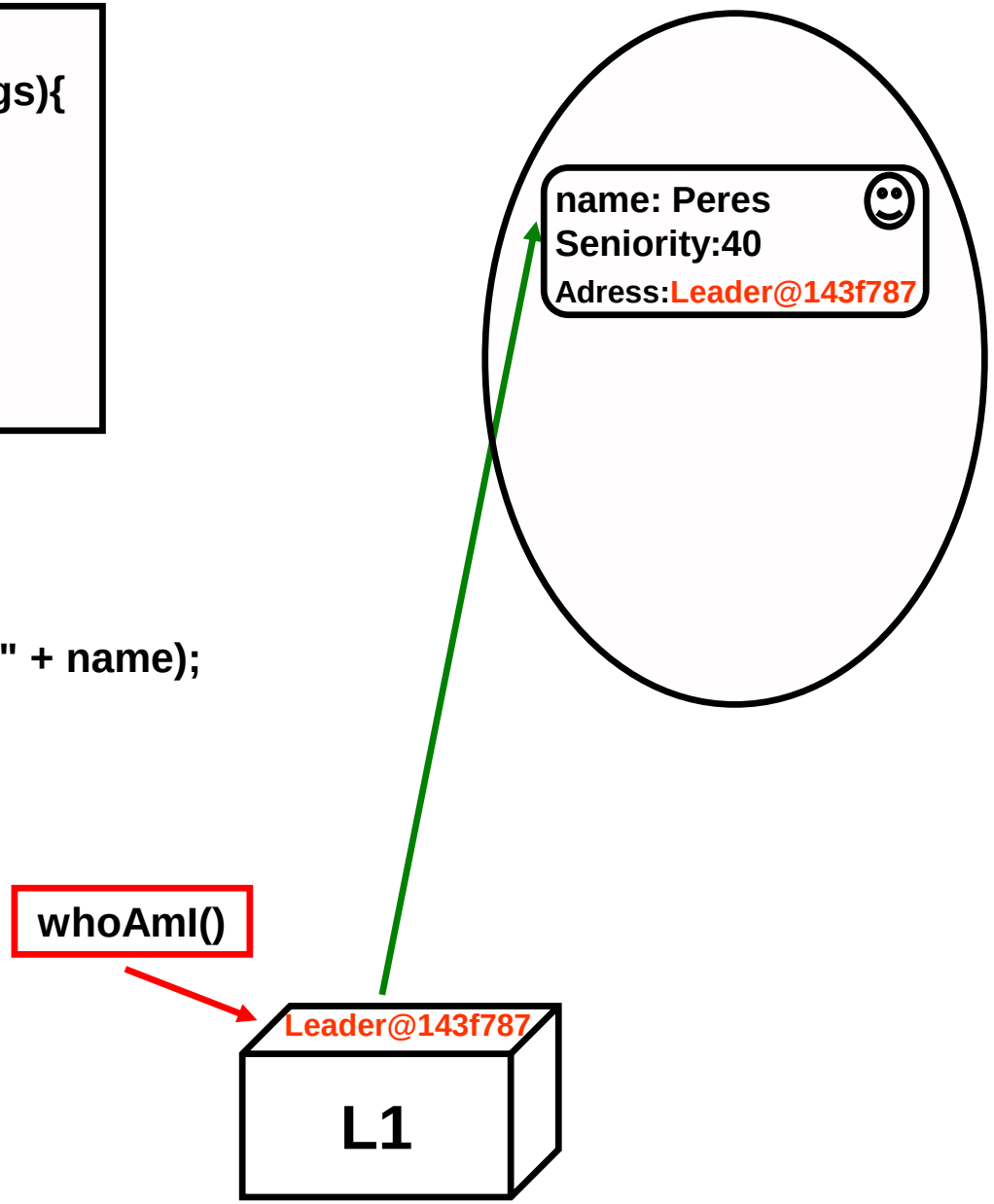


```
public class Tester {
    public static void main(String[ ] args){
        Leader L1;
        L1=new Leader("Peres", 40);
        L1.whoAmI();
        System.out.println(L1);
    }
}
```



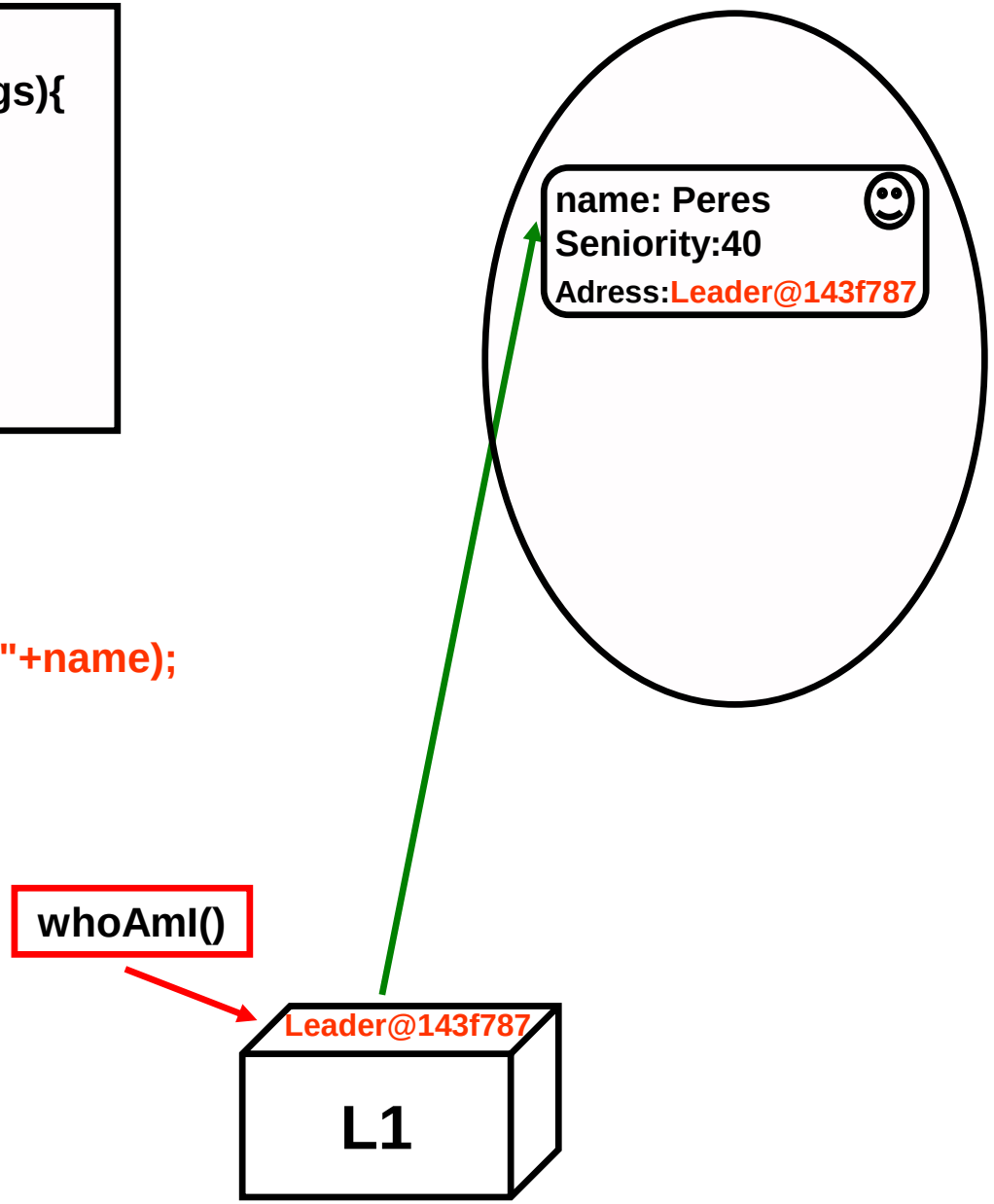
```
public class Tester {
    public static void main(String[ ] args){
        Leader L1;
        L1=new Leader("Peres", 40);
        L1.whoAmI();
        System.out.println(L1);
    }
}
```

```
public void whoAmI(){
        System.out.println("My name is " + name);
}
```



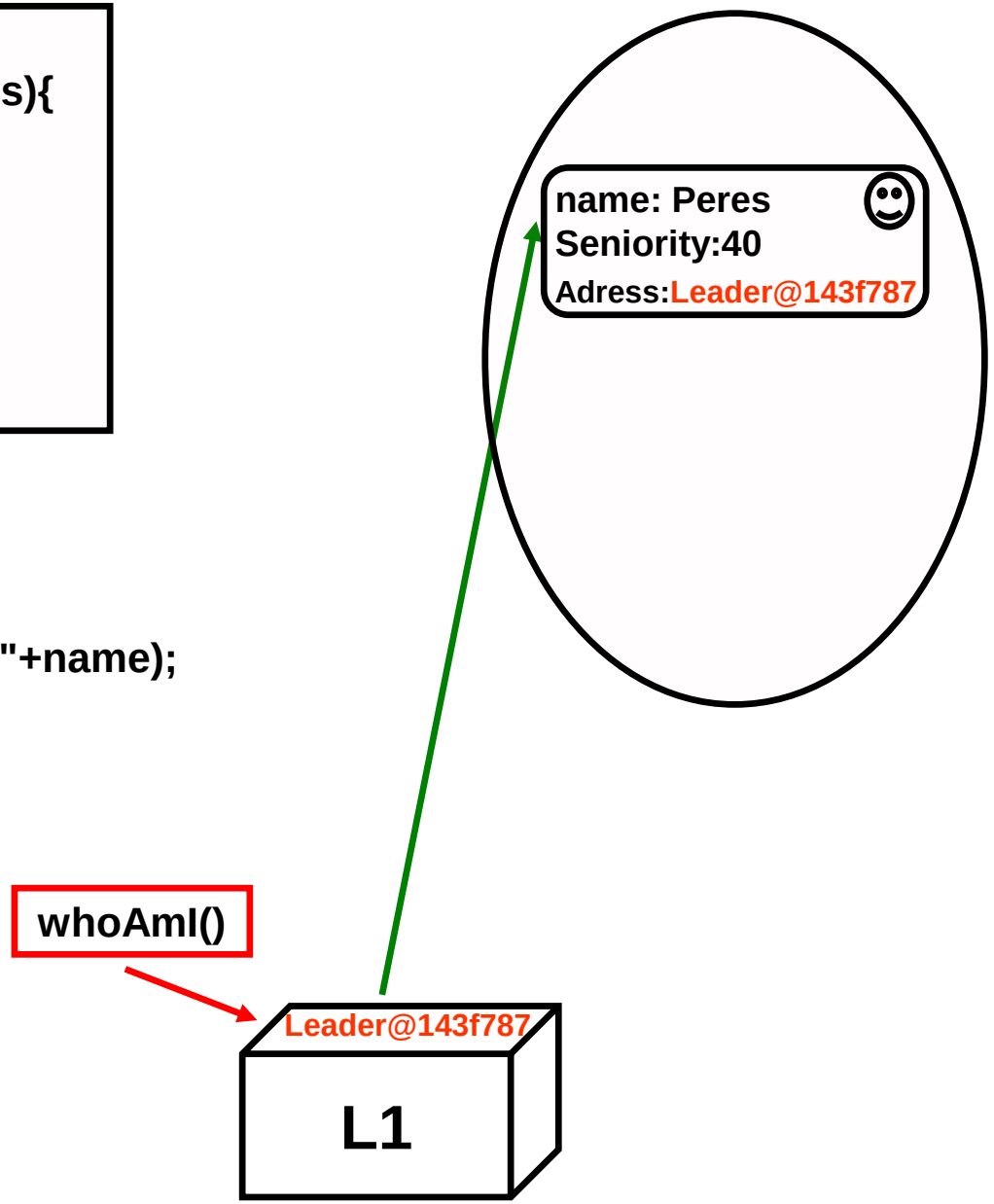
```
public class Tester {
    public static void main(String[ ] args){
        Leader L1;
        L1=new Leader("Peres", 40);
        L1.whoAmI();
        System.out.println(L1);
    }
}
```

```
public void whoAmI() {
        System.out.println("My name is "+name);
}
```



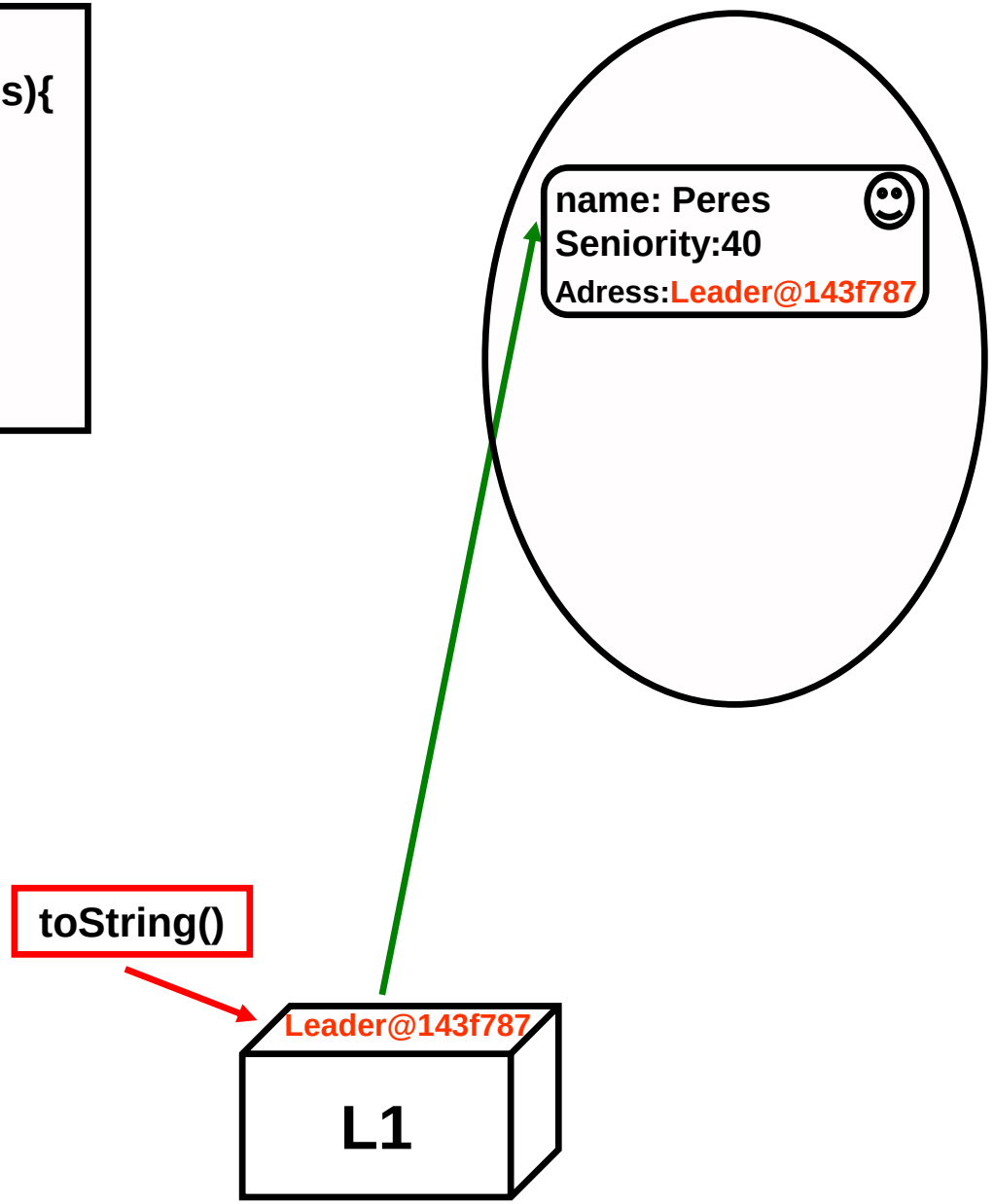
```
public class Tester {
    public static void main(String[] args){
        Leader L1;
        L1=new Leader("Peres", 40);
        L1.whoAmI();
        System.out.println(L1);
    }
}
```

```
public void whoAmI(){
        System.out.println("My name is "+name);
}
```



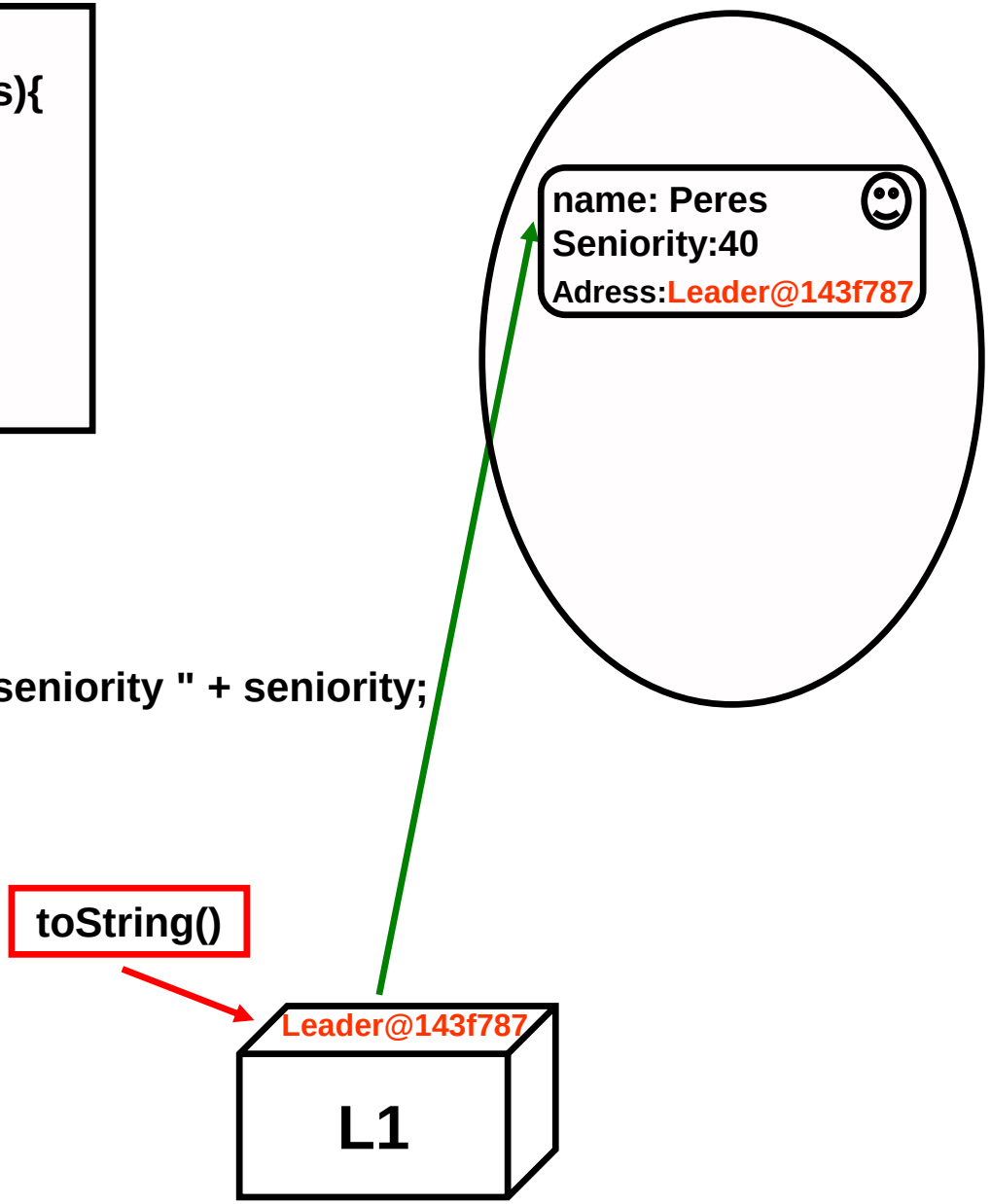
```
public class Tester {
    public static void main(String[] args){
        Leader L1;
        L1=new Leader("Peres", 40);
        L1.whoAml();
        System.out.println(L1);
    }
}
```

איזה toString() יופעל?
של Object , Person
או של Leader ?
תשובה: השיטה העדכנית ביותר
היא זו שתופעל ובמקרה הזה
השיטה שהוגדרה ב Leader



```
public class Tester {
    public static void main(String[] args){
        Leader L1;
        L1=new Leader("Peres", 40);
        L1.whoAmI();
        System.out.println(L1);
    }
}
```

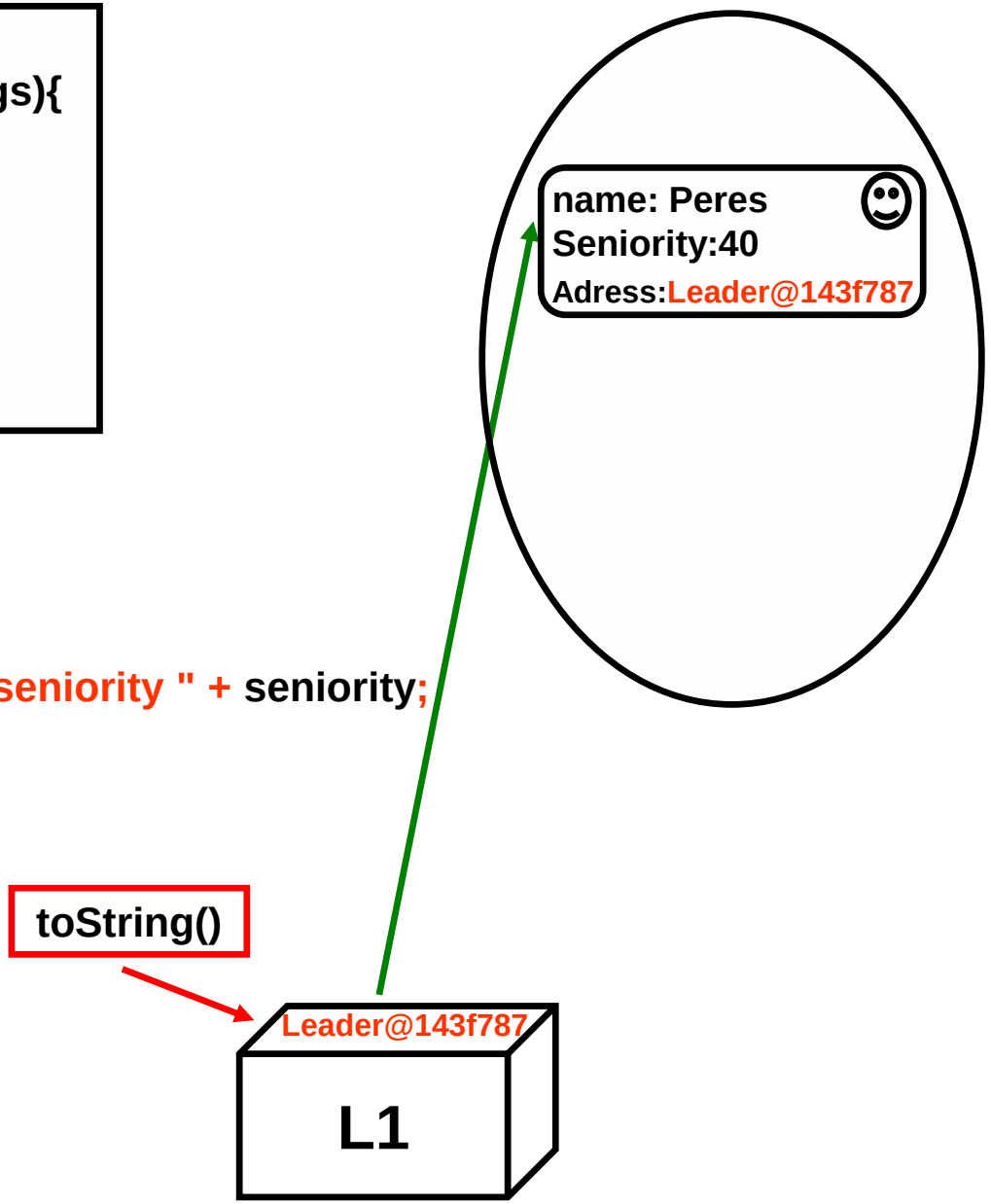
```
public String toString(){
    return "Leader " + name + " with seniority " + seniority;
}
```



```
public class Tester {
    public static void main(String[ ] args){
        Leader L1;
        L1=new Leader("Peres", 40);
        L1.whoAml();
        System.out.println(L1);
    }
}
```

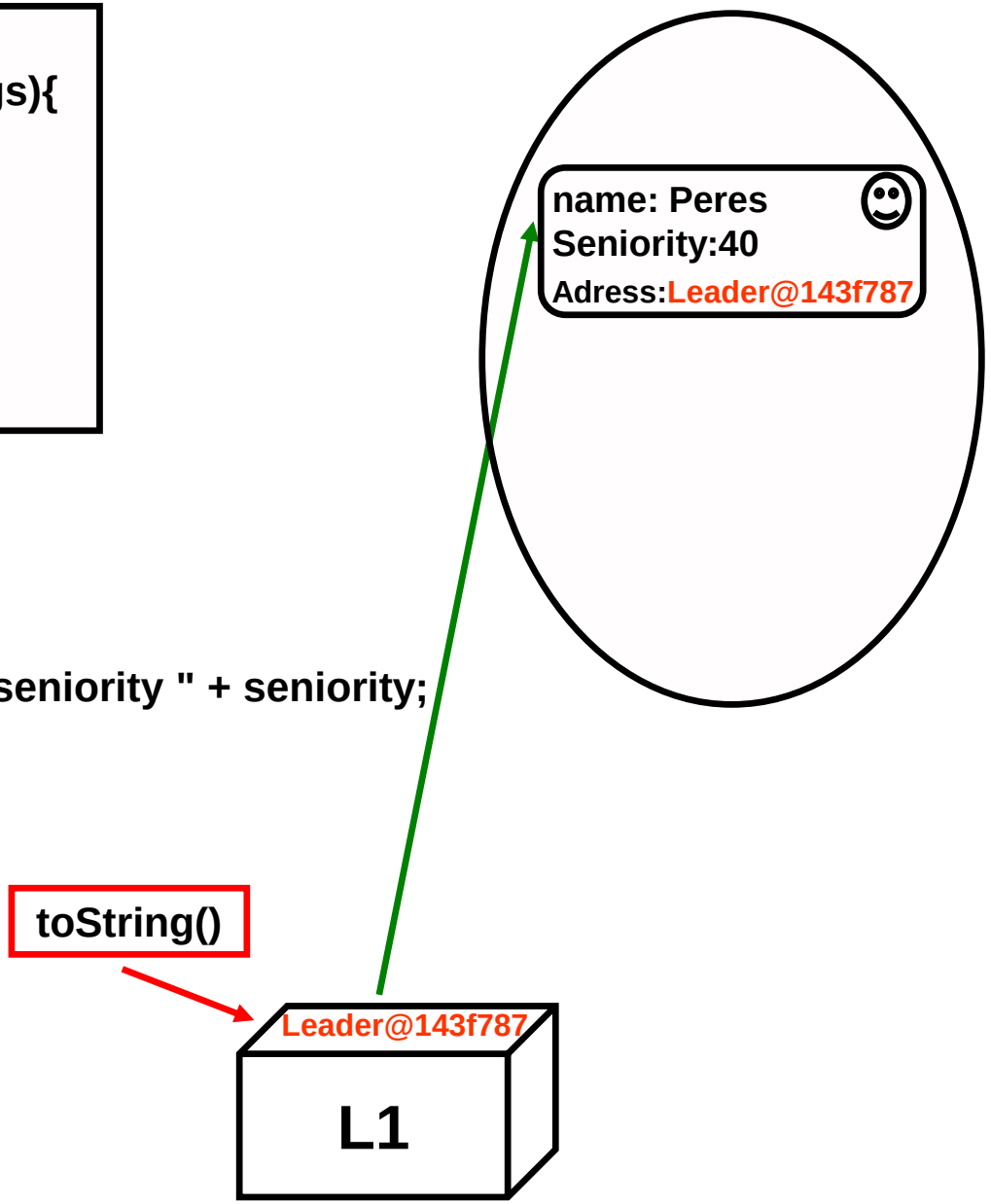
```
public String toString(){
    return "Leader " + name + " with seniority " + seniority;
}
```

- אם נחליף את name ב super.name מה נקבל ?
- אם נחליף את name ב super.toString() מה נקבל?

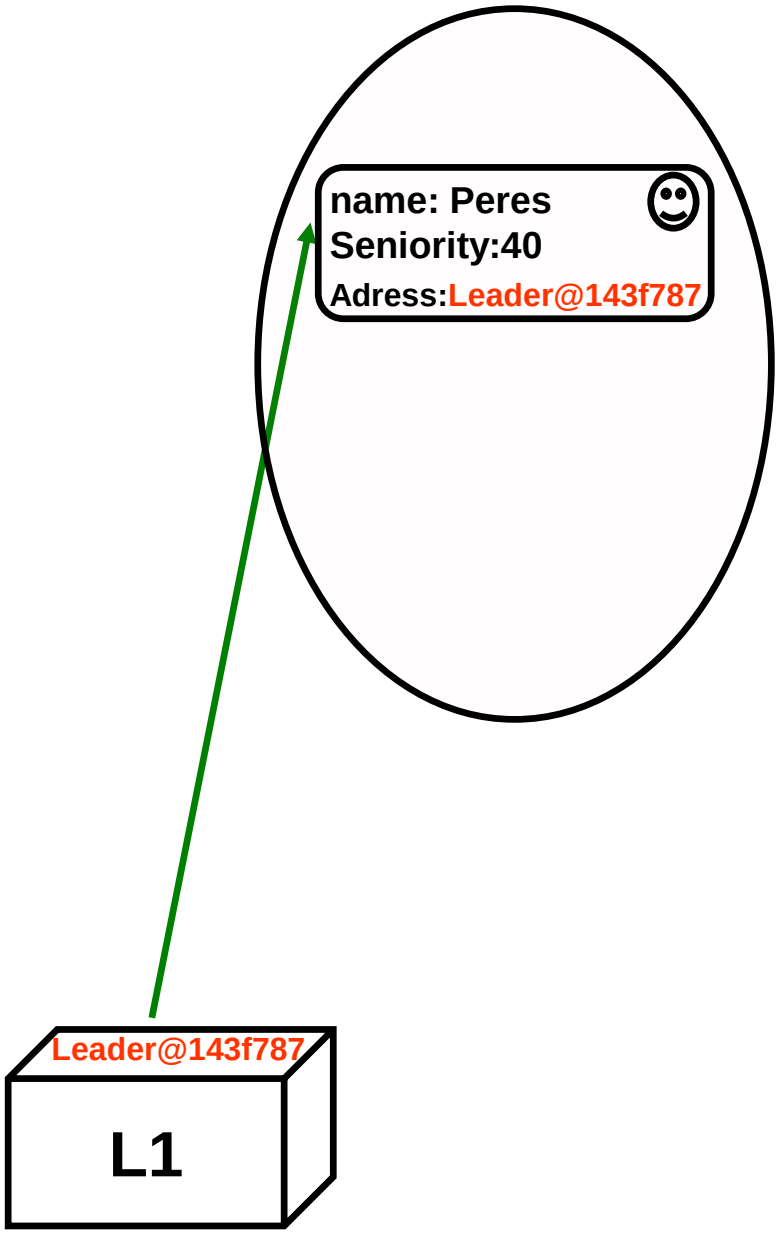


```
public class Tester {
    public static void main(String[] args){
        Leader L1;
        L1=new Leader("Peres", 40);
        L1.whoAmI();
        System.out.println(L1);
    }
}
```

```
public String toString(){
    return "Leader " + name + " with seniority " + seniority;
}
```



```
public class Tester {
    public static void main(String[] args){
        Leader L1;
        L1=new Leader("Peres", 40);
        L1.whoAml();
        System.out.println(L1);
    }
}
```

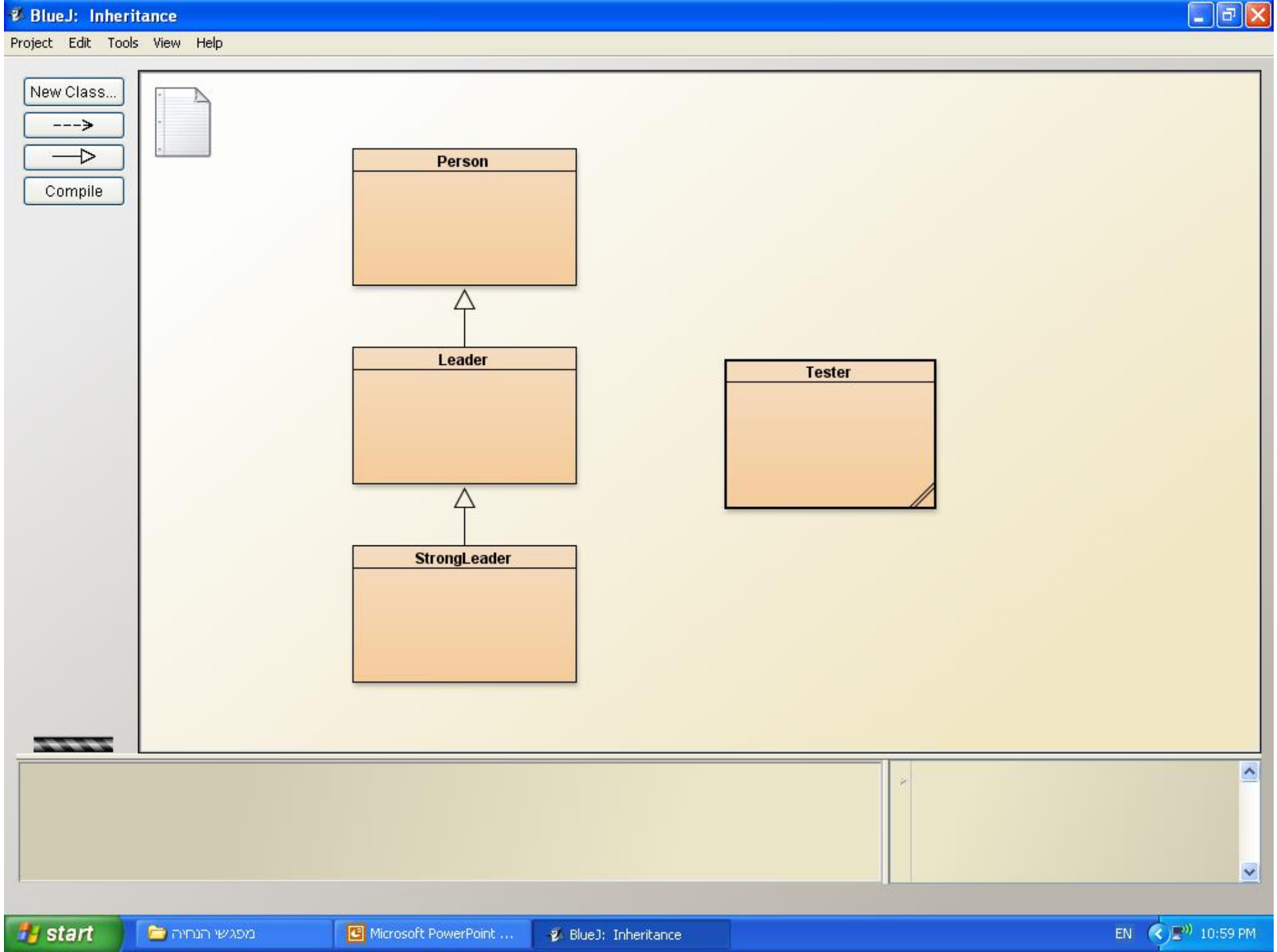


בדומה לתרגיל הקודם נגדיר הפעם את המחלקה StrongLeader אשר יורשת מ Leader. כמו כן מצורף בעמוד הבא הטסטר שרוצים לבצע עליו מעקב.

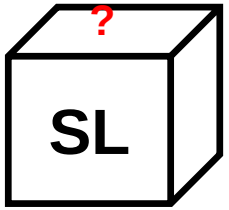
```
public class StrongLeader extends Leader{
   private int strength;

   public StrongLeader(String name , int seniority , int strength) {
    super (name,seniority);
    this.strength=strength;
    System.out.println("Strong Leader constructor has been generated");
   }
   public String toString() {
      return super.toString() + " and strength " + strength;
   }
}
```

```
public class Tester {
    public static void main(String[] args){
        StrongLeader SL=new StrongLeader("Peres",40,3);
        SL.whoAmI();
        System.out.println(SL);
    }
}
```

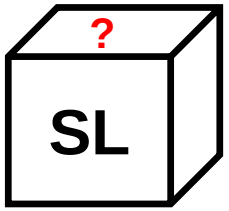
```
Output:
Person constructor has been generated
Leader constructor has been generated
Strong Leader constructor has been generated
My name is Peres
Leader Peres with seniority 40 and strength 3
```


BlueJ: Inheritance
Project Edit Tools View Help
New Class...
Compile
Person
Leader
StrongLeader
Tester
start
Microsoft PowerPoint ...
BlueJ: Inheritance
EN
10:59 PM

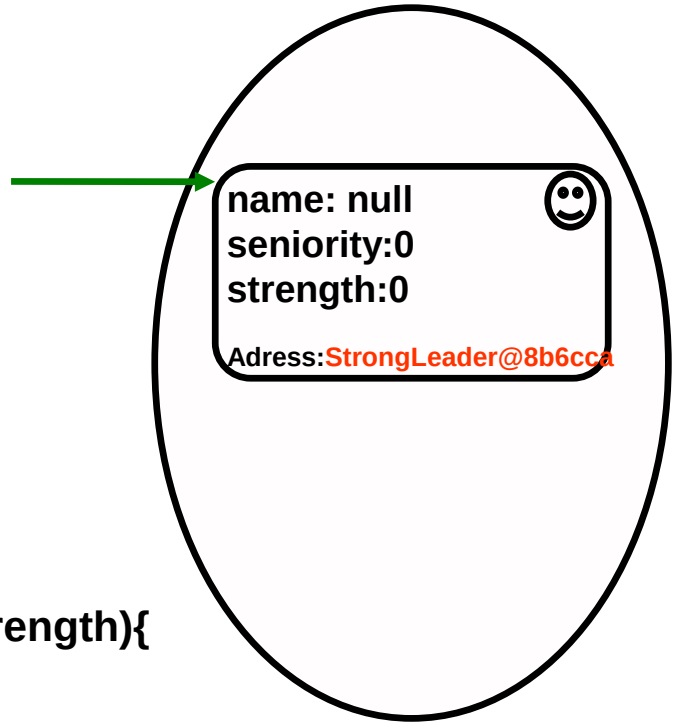
```
public class Tester {
    public static void main(String[] args){
        StrongLeader SL;
        SL=new StrongLeader("Peres",40,3);
        SL.whoAmI();
        System.out.println(SL);
    }
}
```



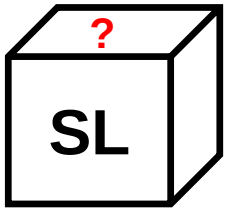
```
public class Tester {
    public static void main(String [ ] args){
        StrongLeader SL;
        SL=new StrongLeader("Peres",40,3);
        SL.whoAmI();
        System.out.println(SL);
    }
}
```



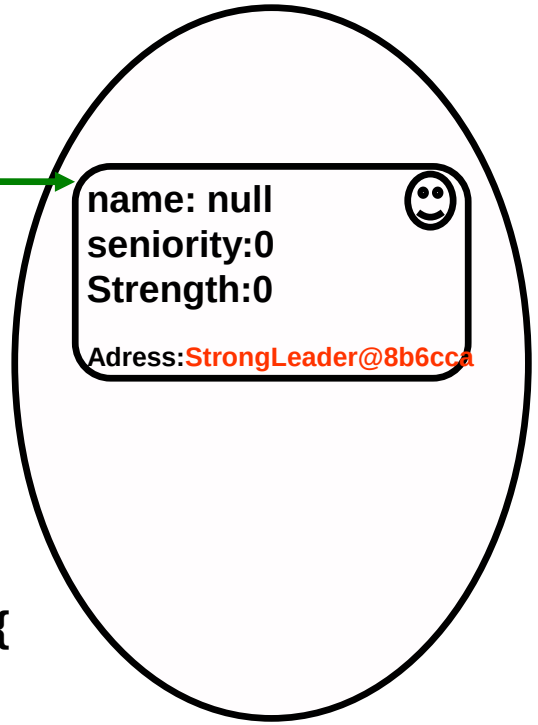
```
public class Tester {
    public static void main(String[ ] args){
        StrongLeader SL;
        SL=new StrongLeader("Peres",40,3);
        SL.whoAmI();
        System.out.println(SL);
    }
}
```



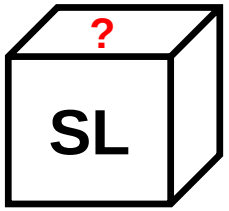

```
public StrongLeader(String name ,int seniority ,int strength){
    super(name,seniority);
    this.strength=strength;
    System.out.println("Strong Leader constructor has been generated");
}
```



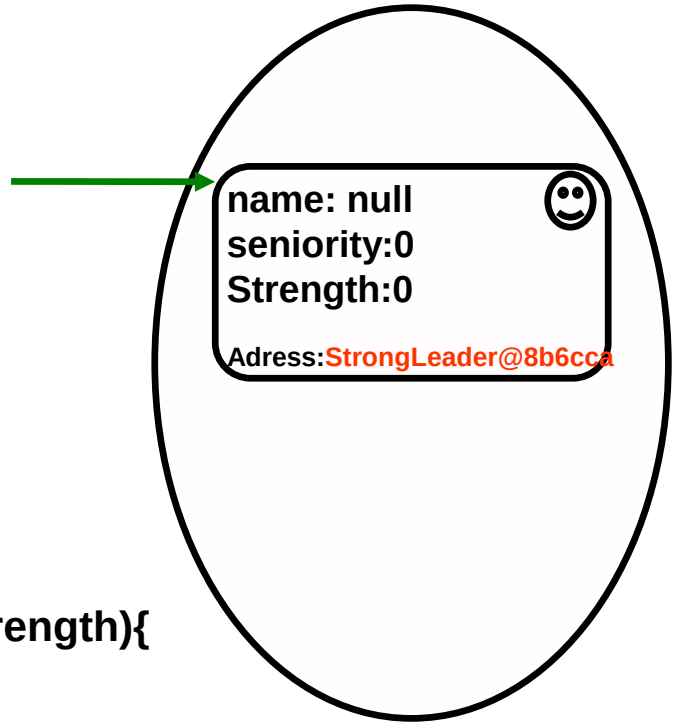
```
public class Tester {
    public static void main(String[ ] args){
        StrongLeader SL;
        SL=new StrongLeader("Peres",40,3);
        SL.whoAmI();
        System.out.println(SL);
    }
}
```



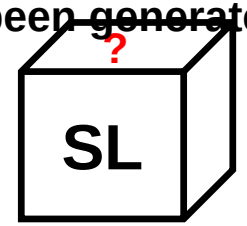

```
public StrongLeader(String name ,int seniority ,int strength){
    super(name,seniority);
    this.strength=strength;
    System.out.println("Strong Leader constructor has been generated");
}
```



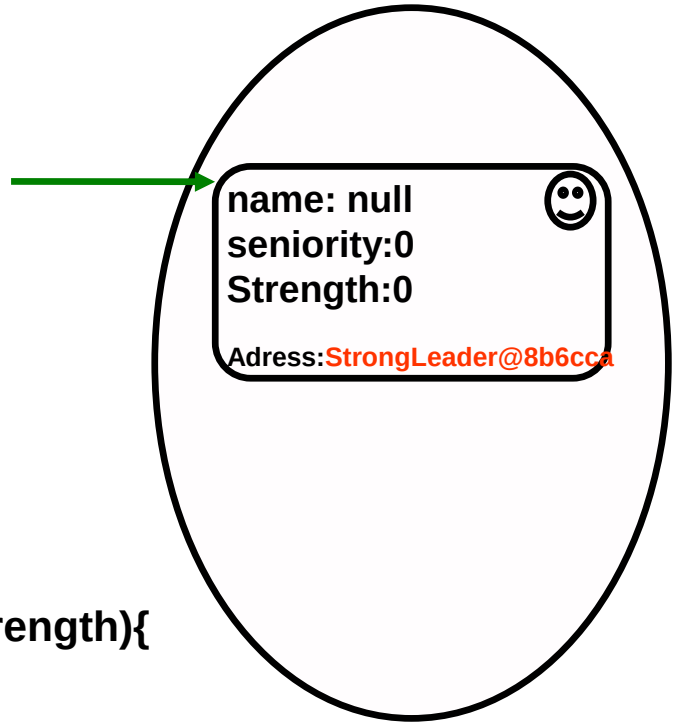
```
public class Tester {
    public static void main(String[] args){
        StrongLeader SL;
        SL=new StrongLeader("Peres",40,3);
        SL.whoAmI();
        System.out.println(SL);
    }
}
```



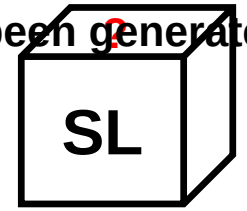

```
public StrongLeader(String name ,int seniority ,int strength){
    super(name,seniority);
    this.strength=strength;
    System.out.println("Strong Leader constructor has been generated");
}
public Leader(String name,int seniority){
    super(name);
    this.seniority=seniority;
    System.out.println("Leader constructor has been generated");
}
```



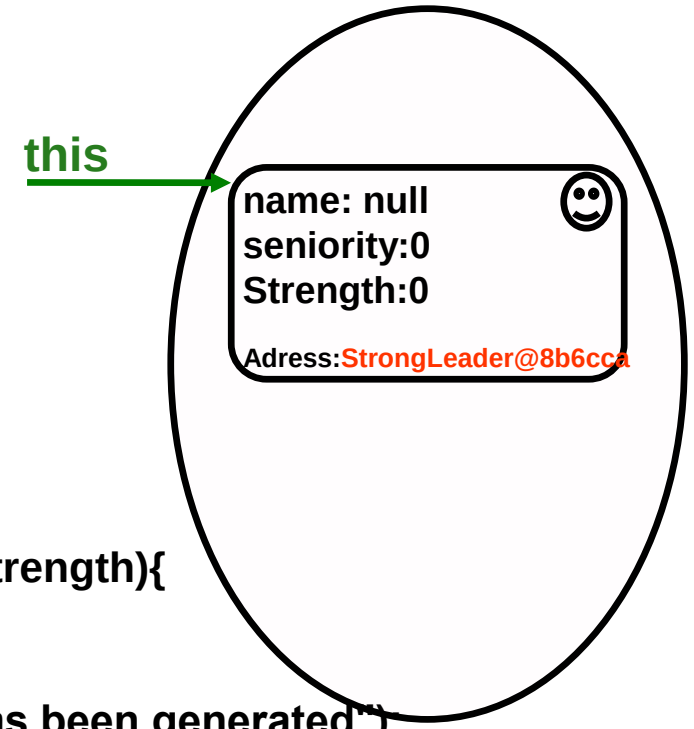
```
public class Tester {
    public static void main(String[] args){
        StrongLeader SL;
        SL=new StrongLeader("Peres",40,3);
        SL.whoAml();
        System.out.println(SL);
    }
}
```



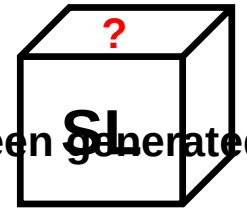

```
public StrongLeader(String name ,int seniority ,int strength){
    super(name,seniority);
    this.strength=strength;
    System.out.println("Strong Leader constructor has been generated");
    }
public Leader(String name,int seniority){
        super(name);
        this.seniority=seniority;
        System.out.println("Leader constructor has been generated");
}
```



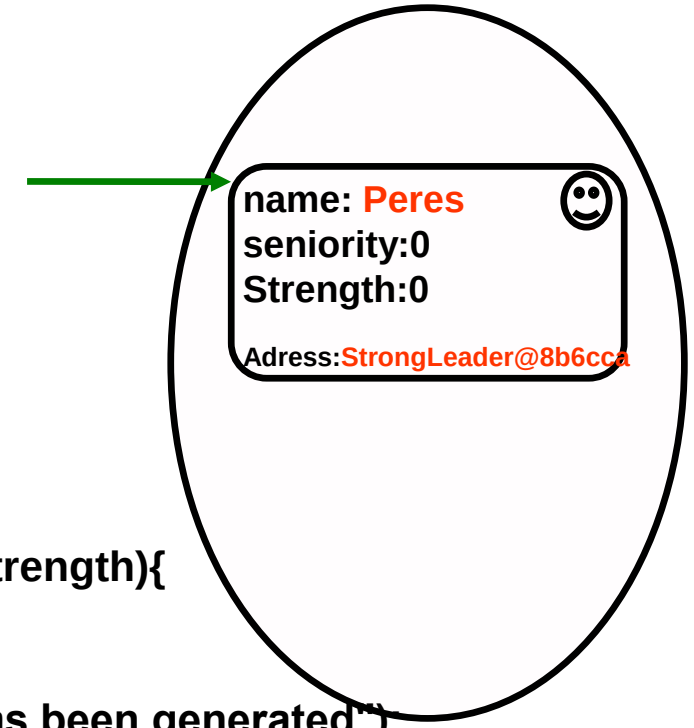
```
public class Tester {
   public static void main(String[ ] args){
      StrongLeader SL;
      SL=new StrongLeader("Peres",40,3);
      SL.whoAmI();
      System.out.println(SL);
   }
}
```



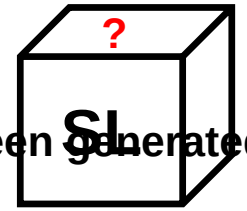

```
public StrongLeader(String name ,int seniority ,int strength){
   super(name,seniority);
   this.strength=strength;
   System.out.println("Strong Leader constructor has been generated");
}
public Leader(String name,int seniority){
      super(name);
      this.seniority=seniority;
      System.out.println("Leader constructor has been generated");
}
public Person(String name){
      this.name=name;
      System.out.println("Person construcor has been generated");
   }
```



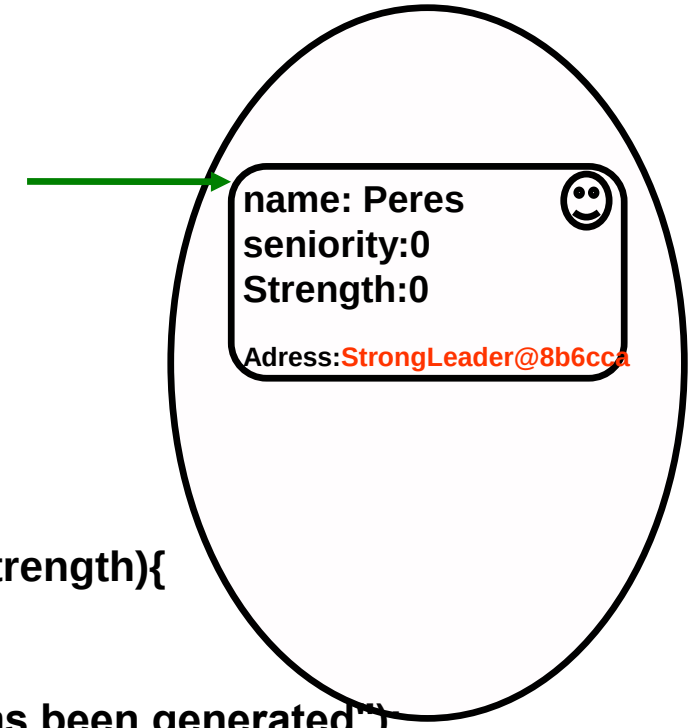
```
public class Tester {
    public static void main(String[ ] args){
        StrongLeader SL;
        SL=new StrongLeader("Peres",40,3);
        SL.whoAml();
        System.out.println(SL);
    }
}
```



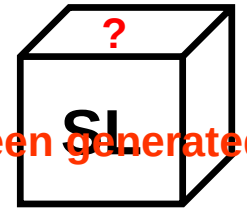

```
public StrongLeader(String name ,int seniority ,int strength){
    super(name,seniority);
    this.strength=strength;
    System.out.println("Strong Leader constructor has been generated");
}
public Leader(String name,int seniority){
    super(name);
    this.seniority=seniority;
    System.out.println("Leader constructor has been generated");
}
public Person(String name){
    this.name=name;
    System.out.println("Person construcor has been generated");
}
```



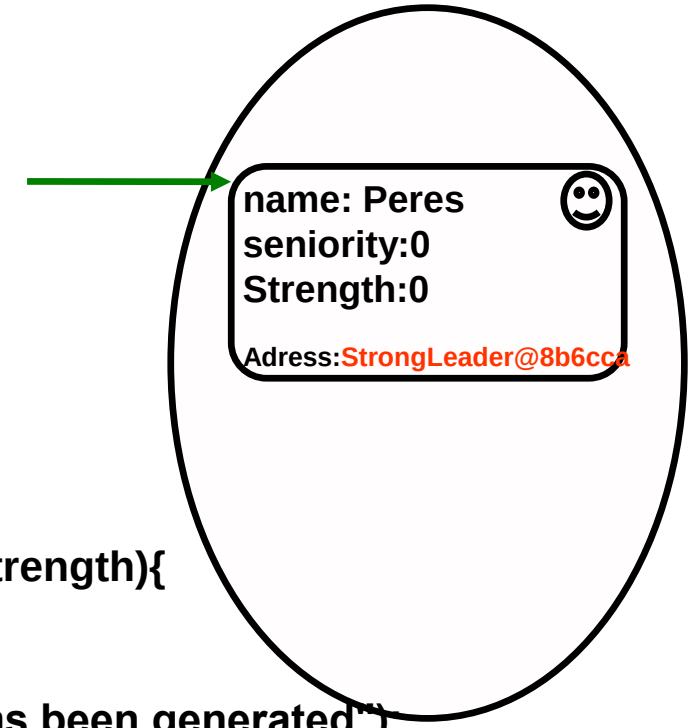
```
public class Tester {
   public static void main(String[ ] args){
      StrongLeader SL;
      SL=new StrongLeader("Peres",40,3);
      SL.whoAmI();
      System.out.println(SL);
   }
}
```



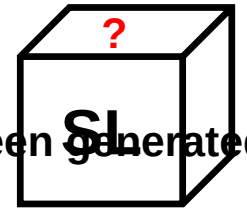

```
public StrongLeader(String name ,int seniority ,int strength){
    super(name,seniority);
    this.strength=strength;
    System.out.println("Strong Leader constructor has been generated");
 }
public Leader(String name,int seniority){
      super(name);
      this.seniority=seniority;
      System.out.println("Leader constructor has been generated");
}
public Person(String name){
      this.name=name;
      System.out.println("Person construcor has been generated!");
   }
```



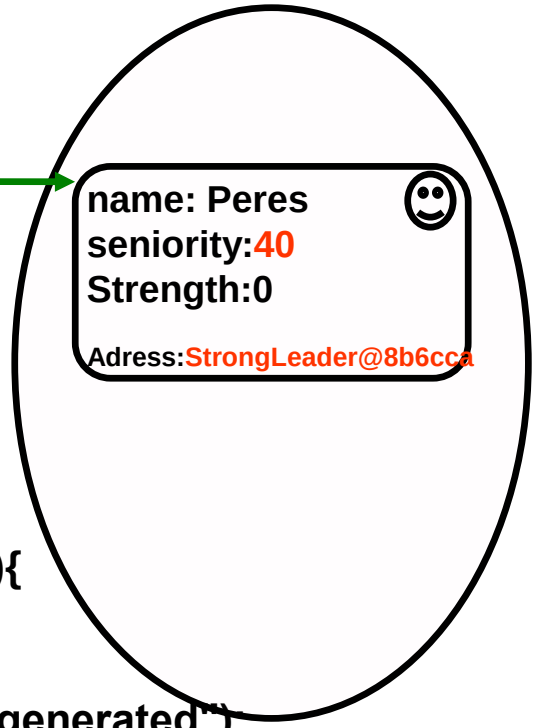
```
public class Tester {
   public static void main(String[ ] args){
      StrongLeader SL;
      SL=new StrongLeader("Peres",40,3);
      SL.whoAmI();
      System.out.println(SL);
   }
}
```



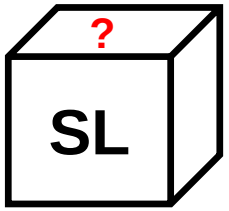

```
public StrongLeader(String name ,int seniority ,int strength){
    super(name,seniority);
    this.strength=strength;
    System.out.println("Strong Leader constructor has been generated");
 }
public Leader(String name,int seniority){
      super(name);
      this.seniority=seniority;
      System.out.println("Leader constructor has been generated");
}
public Person(String name){
      this.name=name;
      System.out.println("Person construcor has been generated!");
   }
```



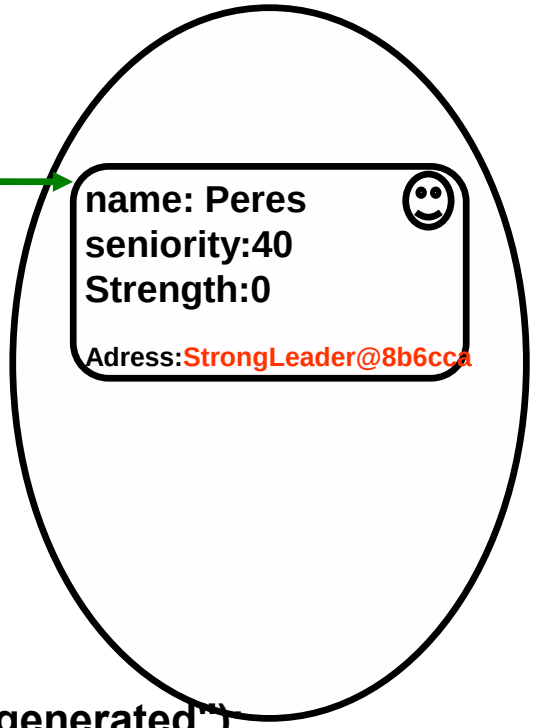
```
public class Tester {
   public static void main(String[] args){
      StrongLeader SL;
      SL=new StrongLeader("Peres",40,3);
      SL.whoAmI();
      System.out.println(SL);
   }
}
```



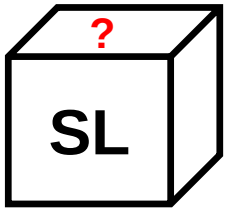

```
public StrongLeader(String name ,int seniority , int strength){
    super(name,seniority);
    this.strength=strength;
    System.out.println("Strong Leader constructor has been generated");
 }
 public Leader(String name , int seniority){
        super(name);
        this.seniority=seniority;
        System.out.println("Leader constructor has been generated");
 }
```



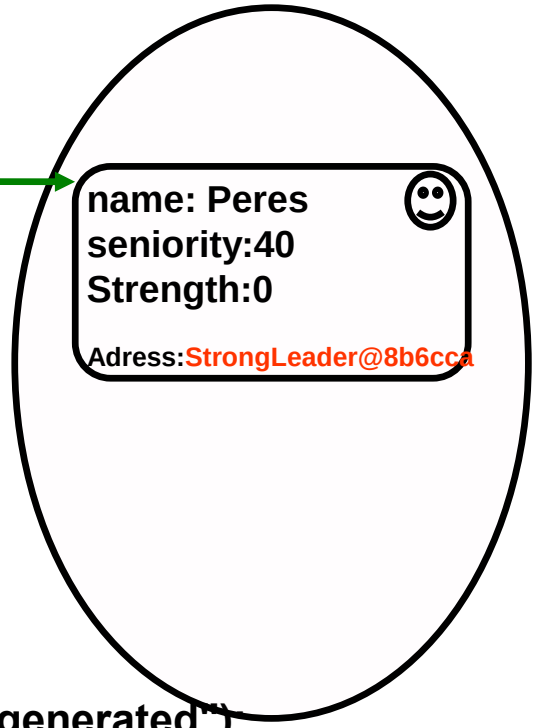
```
public class Tester {
    public static void main(String[ ] args){
        StrongLeader SL;
        SL=new StrongLeader("Peres",40,3);
        SL.whoAml();
        System.out.println(SL);
    }
}
```



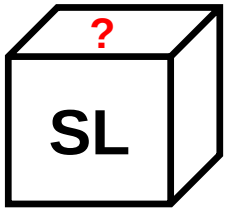

```
public StrongLeader(String name,int seniority,int strength){
    super(name,seniority);
    this.strength=strength;
    System.out.println("Strong Leader constructor has been generated");
}
public Leader(String name,int seniority){
    super(name);
    this.seniority=seniority;
    System.out.println("Leader constructor has been generated");
}
```



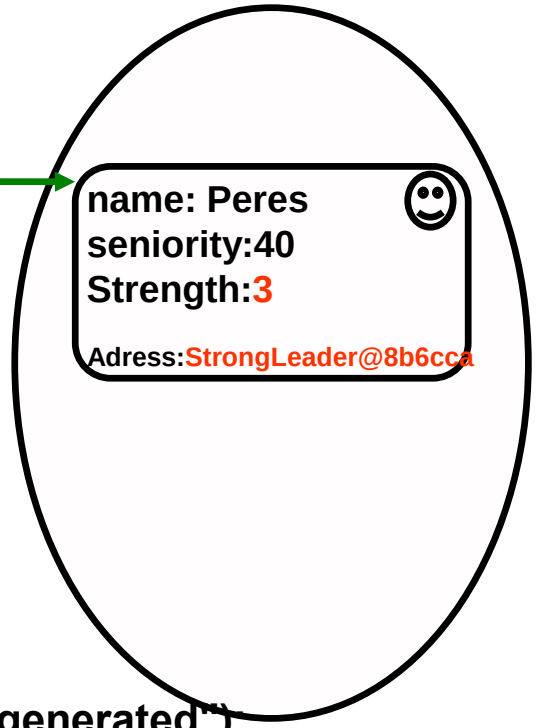
```
public class Tester {
    public static void main(String[] args){
        StrongLeader SL;
        SL=new StrongLeader("Peres",40,3);
        SL.whoAmI();
        System.out.println(SL);
    }
}
```



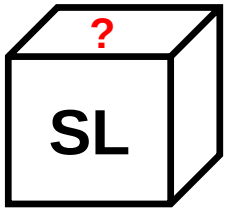

```
public StrongLeader(String name,int seniority,int strength){
    super(name,seniority);
    this.strength=strength;
    System.out.println("Strong Leader constructor has been generated");
}
public Leader(String name,int seniority){
        super(name);
        this.seniority=seniority;
        System.out.println("Leader constructor has been generated");
}
```



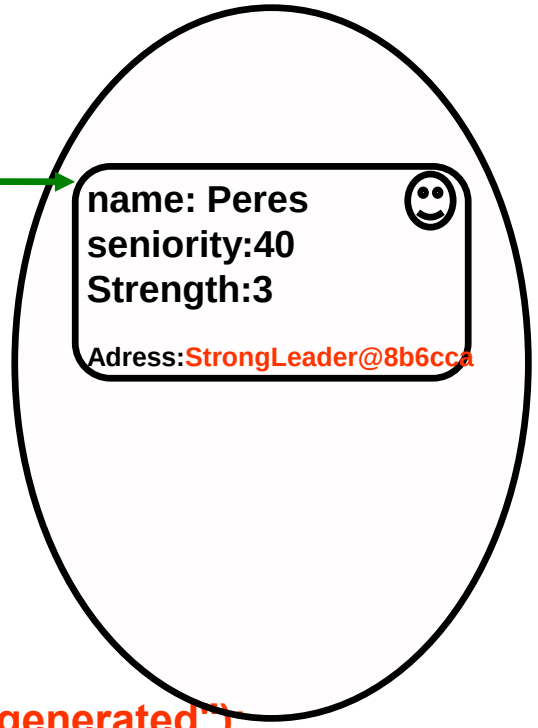
```
public class Tester {
    public static void main(String[] args){
        StrongLeader SL;
        SL=new StrongLeader("Peres",40,3);
        SL.whoAmI();
        System.out.println(SL);
    }
}
```



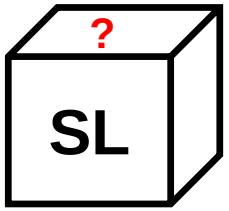

```
public StrongLeader(String name,int seniority,int strength){
    super(name,seniority);
    this.strength=strength;
    System.out.println("Strong Leader constructor has been generated");
}
```



```
public class Tester {
    public static void main(String[ ] args){
        StrongLeader SL;
        SL=new StrongLeader("Peres",40,3);
        SL.whoAml();
        System.out.println(SL);
    }
}
```

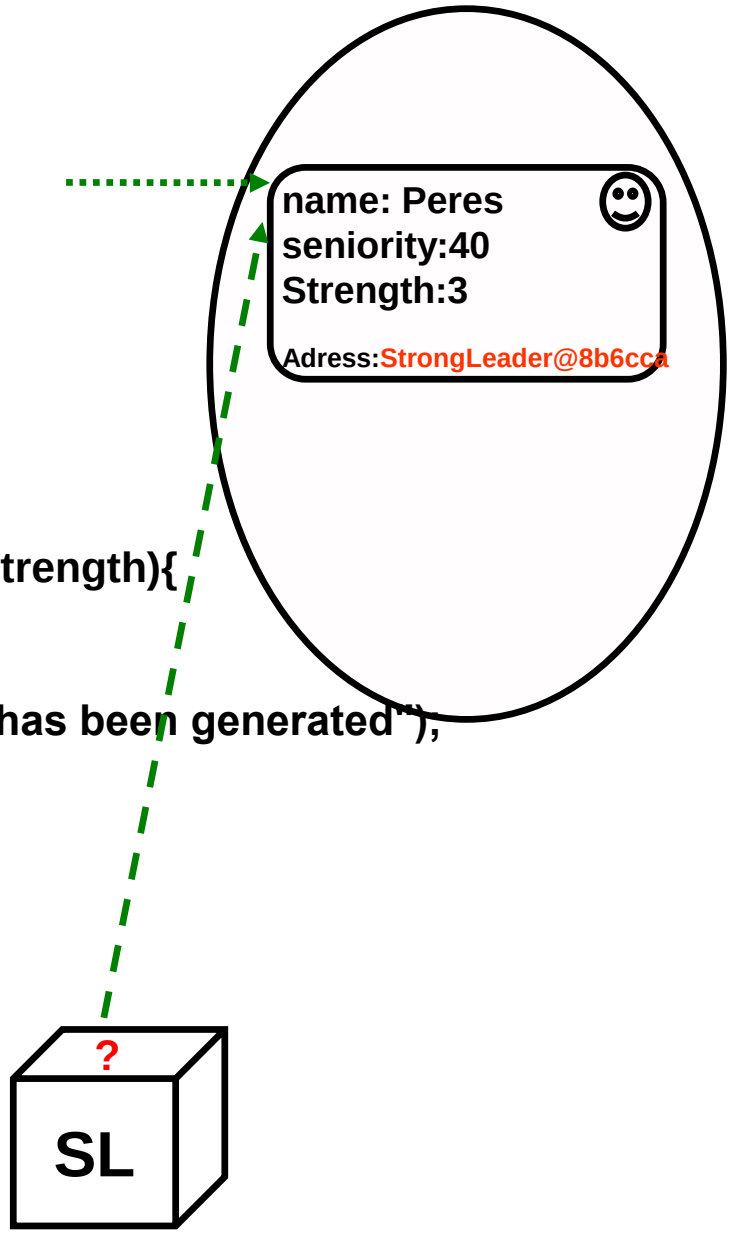



```
public StrongLeader(String name,int seniority,int strength){
    super(name,seniority);
    this.strength=strength;
    System.out.println("Strong Leader constructor has been generated");
}
```

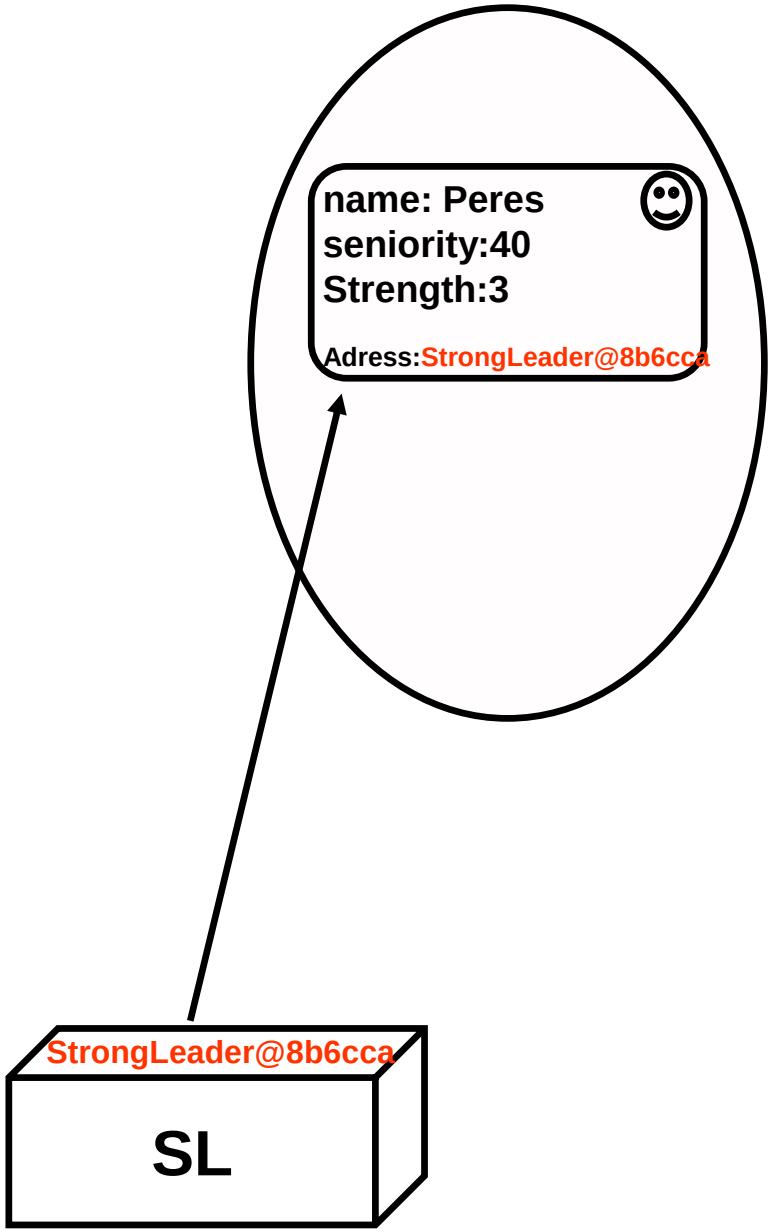


```
public class Tester {
    public static void main(String[ ] args){
        StrongLeader SL;
        SL=new StrongLeader("Peres",40,3);
        SL.whoAml();
        System.out.println(SL);
    }
}
```

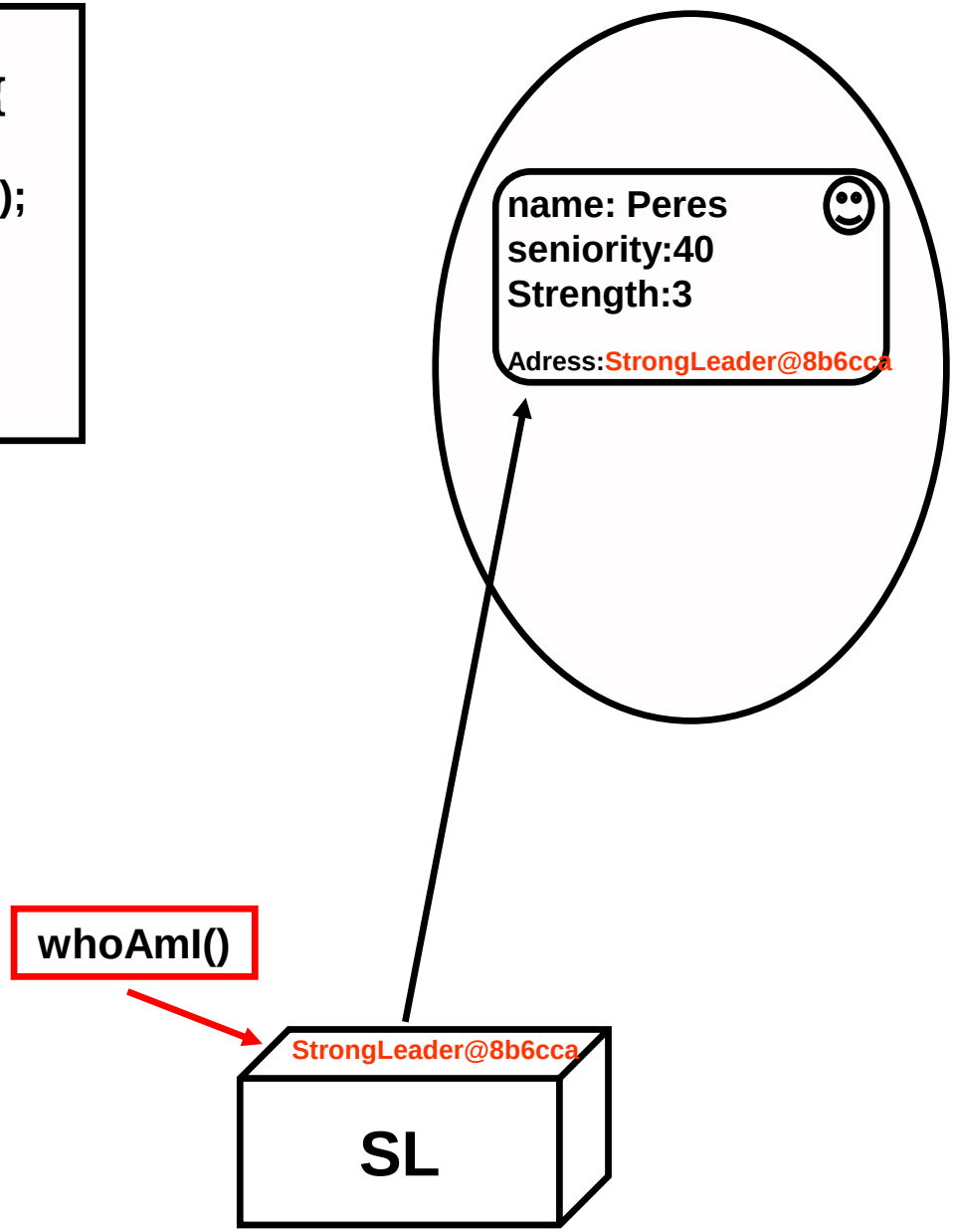
```
public StrongLeader(String name,int seniority,int strength){
    super(name,seniority);
    this.strength=strength;
    System.out.println("Strong Leader constructor has been generated");
}
```



```
public class Tester {
    public static void main(String[ ] args){
        StrongLeader SL;
        SL=new StrongLeader("Peres",40,3);
        SL.whoAmI();
        System.out.println(SL);
    }
}
```

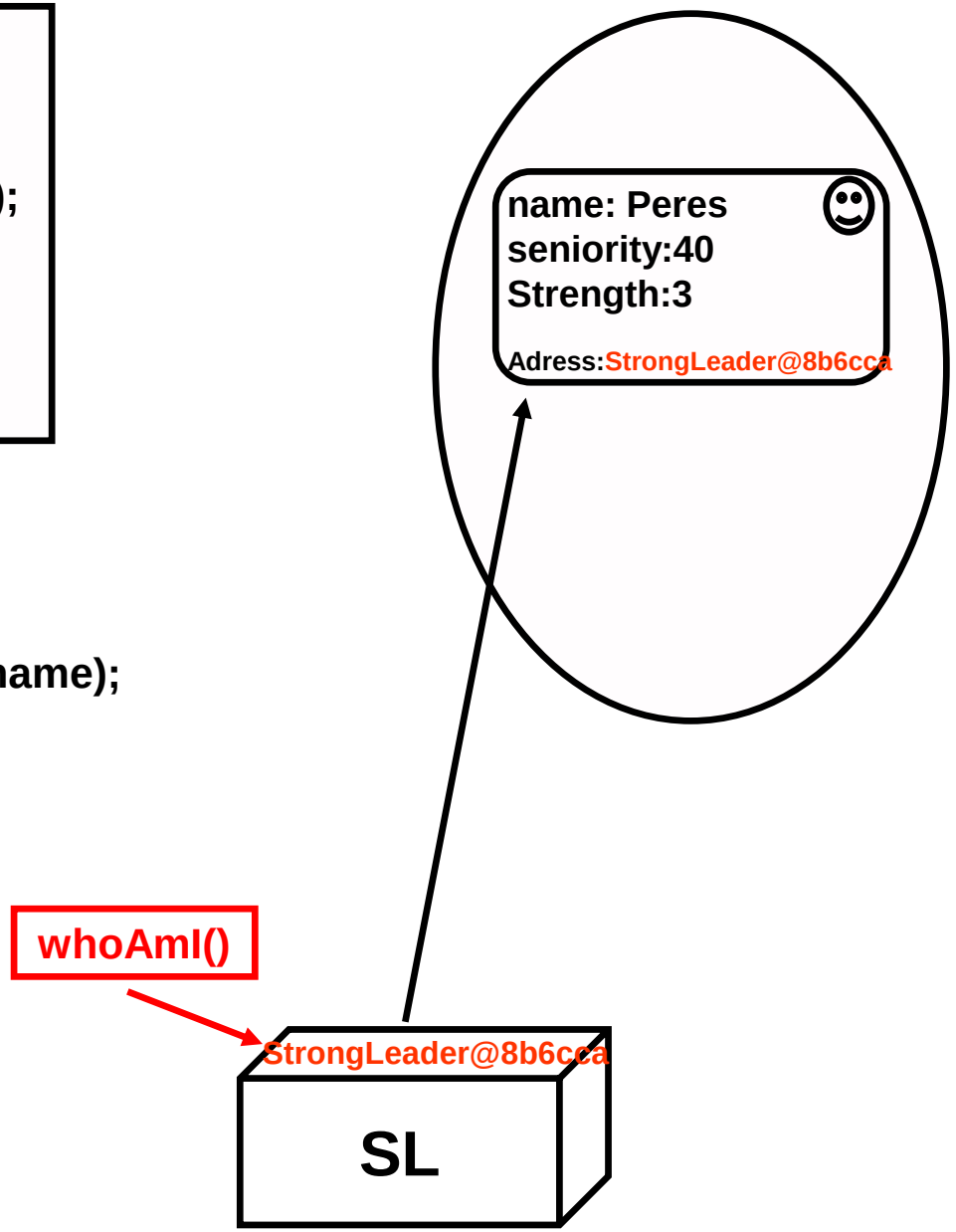


```
public class Tester {
    public static void main(String[ ] args){
        StrongLeader SL;
        SL=new StrongLeader("Peres",40,3);
        SL.whoAmI();
        System.out.println(SL);
    }
}
```



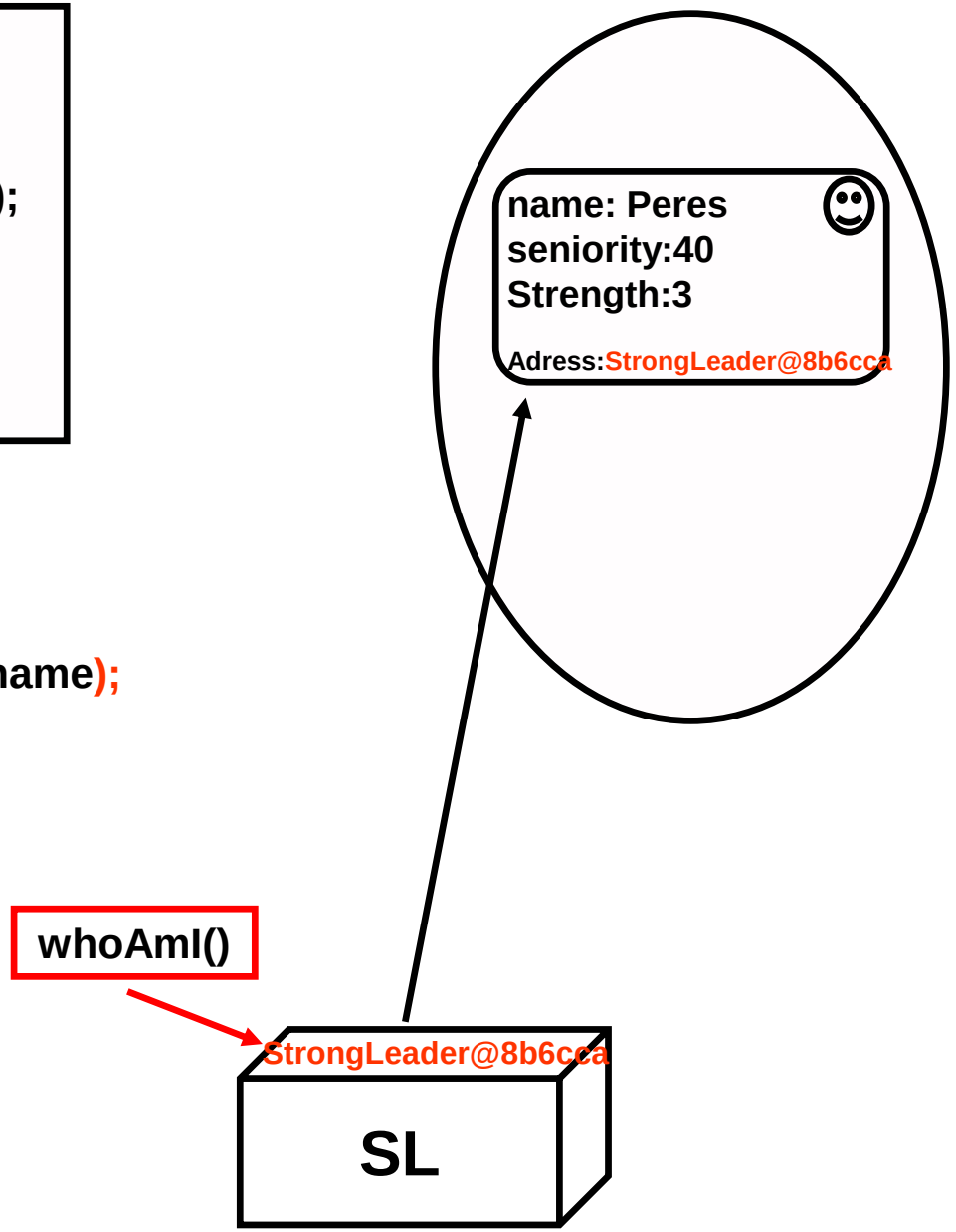
```
public class Tester {
    public static void main(String[ ] args){
        StrongLeader SL;
        SL=new StrongLeader("Peres",40,3);
        SL.whoAmI();
        System.out.println(SL);
    }
}
```

```
public void whoAmI(){
        System.out.println("My name is "+name);
}
```



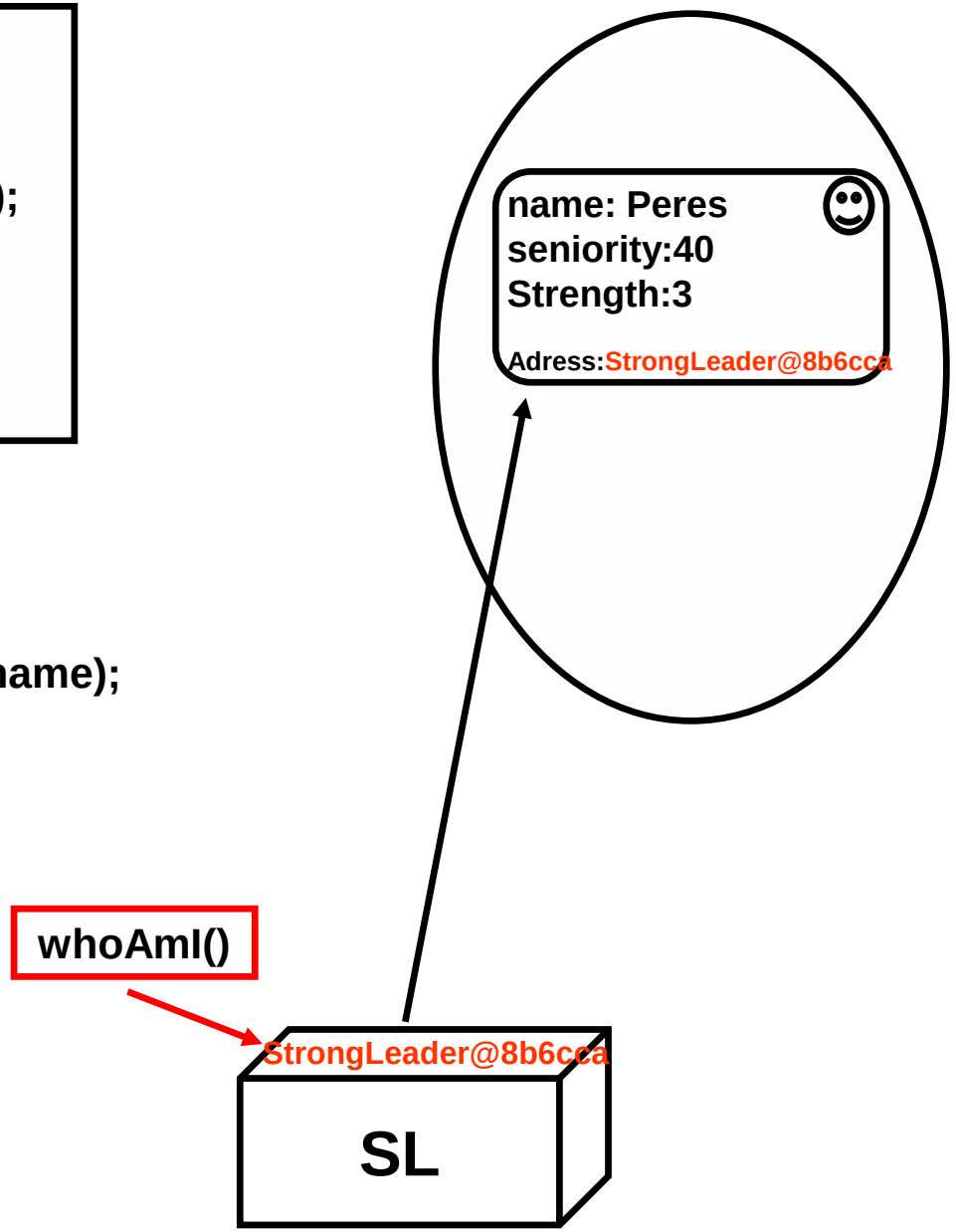
```
public class Tester {
    public static void main(String[ ] args){
        StrongLeader SL;
        SL=new StrongLeader("Peres",40,3);
        SL.whoAmI();
        System.out.println(SL);
    }
}
```

```
public void whoAmI(){
    System.out.println("My name is "+name);
}
```

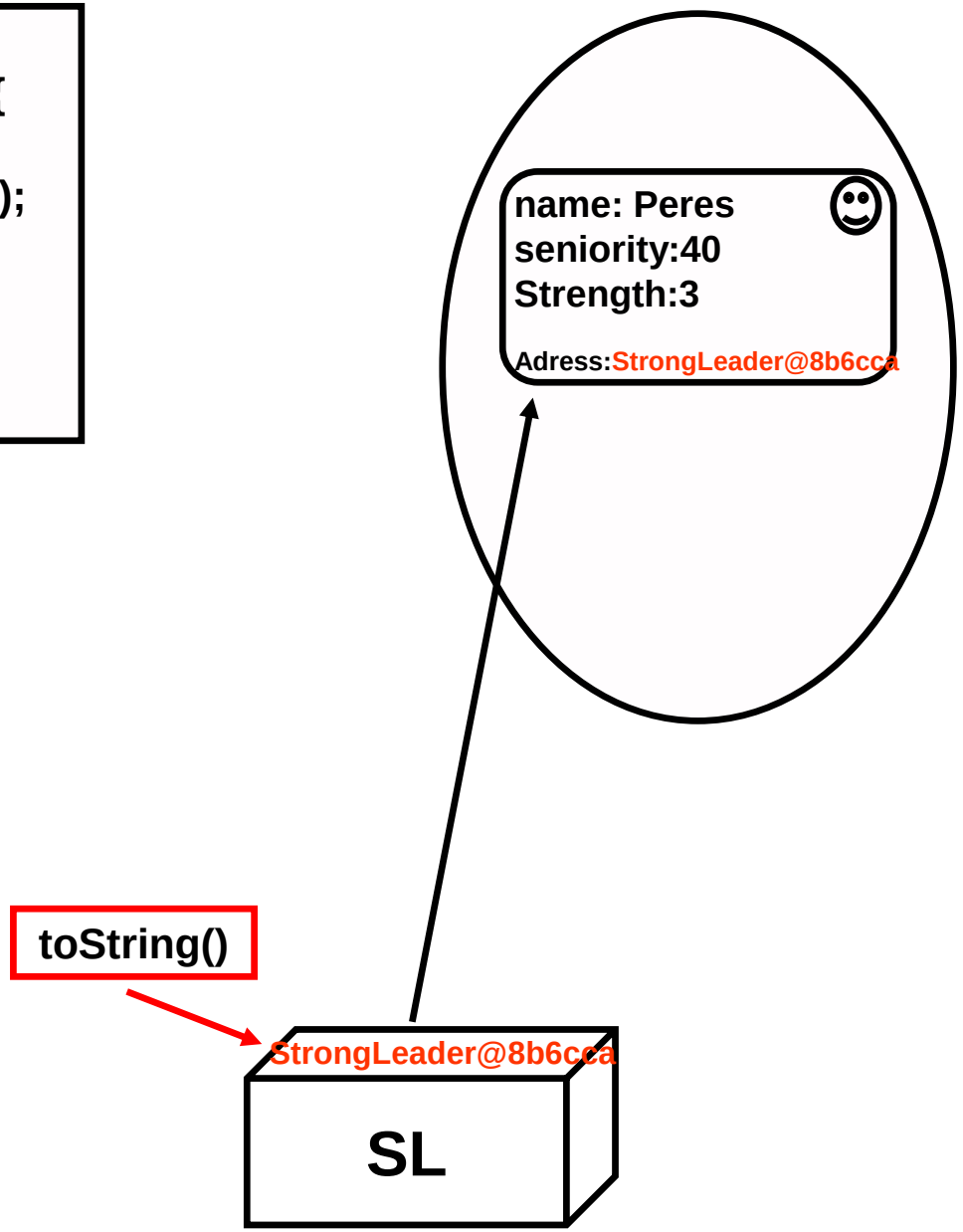


```
public class Tester {
    public static void main(String[ ] args){
        StrongLeader SL;
        SL=new StrongLeader("Peres",40,3);
        SL.whoAmI();
        System.out.println(SL);
    }
}
```

```
public void whoAmI(){
        System.out.println("My name is "+name);
}
```

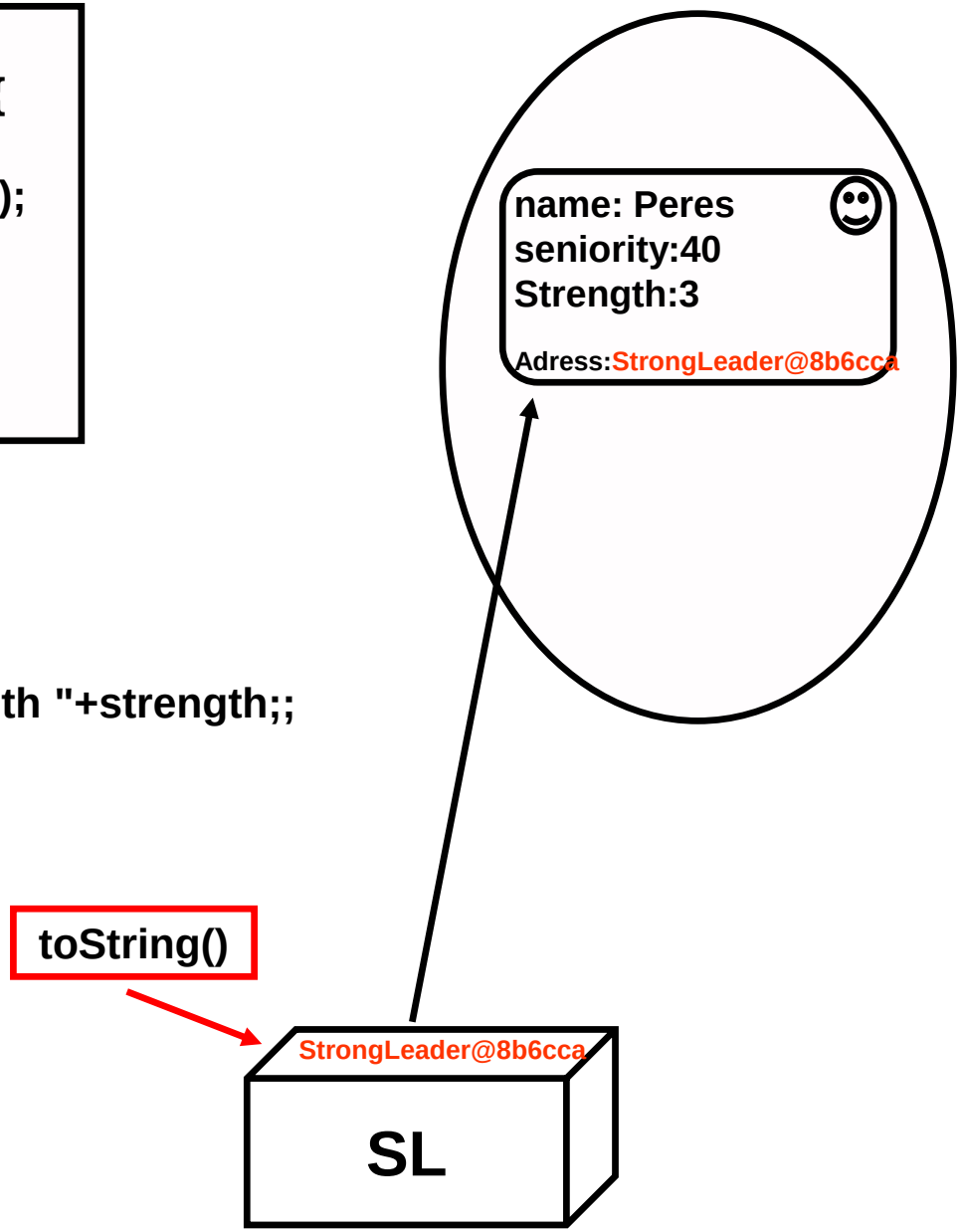


```
public class Tester {
    public static void main(String[ ] args){
        StrongLeader SL;
        SL=new StrongLeader("Peres",40,3);
        SL.whoAml();
        System.out.println(SL);
    }
}
```



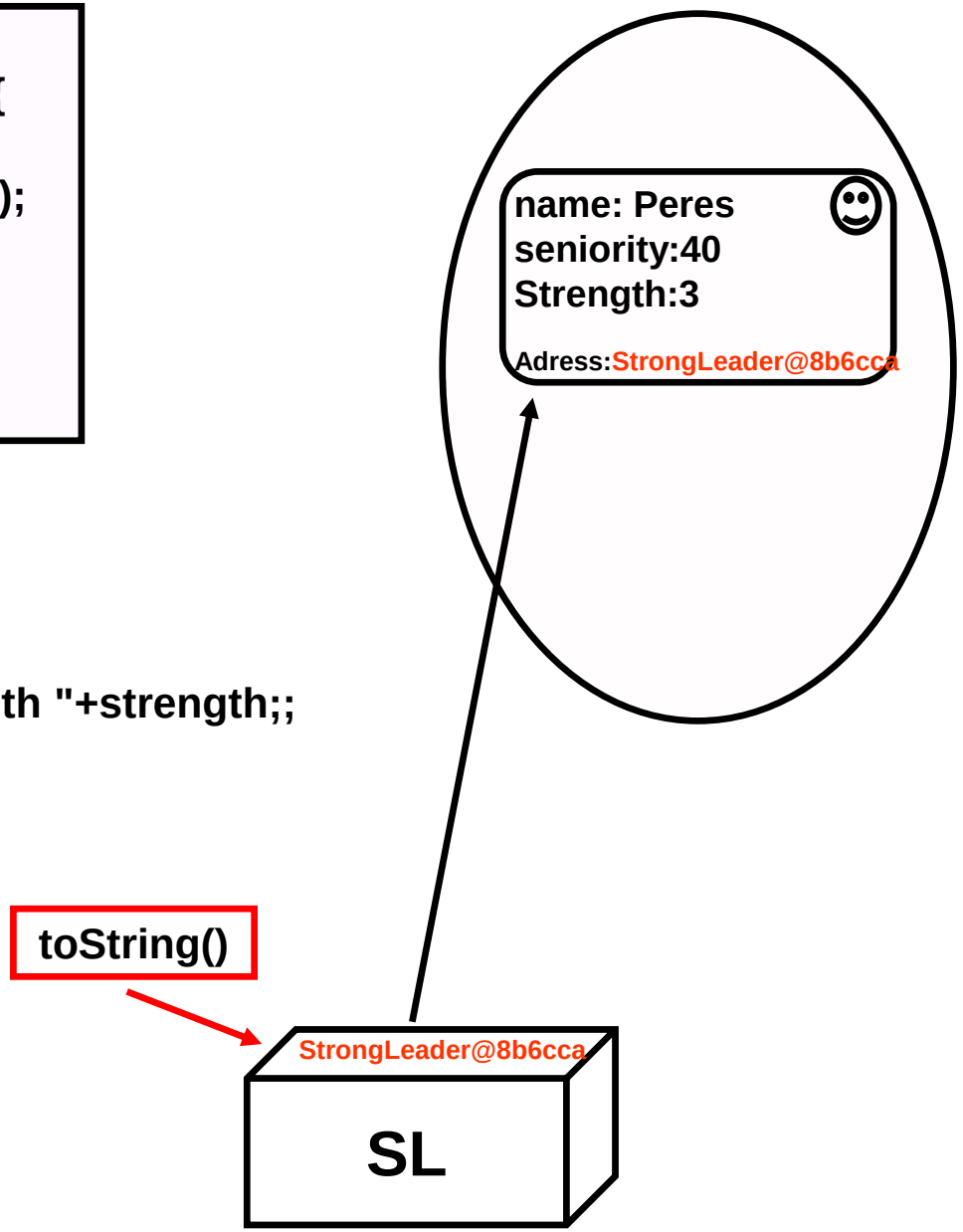
```
public class Tester {
    public static void main(String[ ] args){
        StrongLeader SL;
        SL=new StrongLeader("Peres",40,3);
        SL.whoAml();
        System.out.println(SL);
    }
}
```

```
public String toString(){
        return super.toString()+" and stringth "+strength;;
  }
```



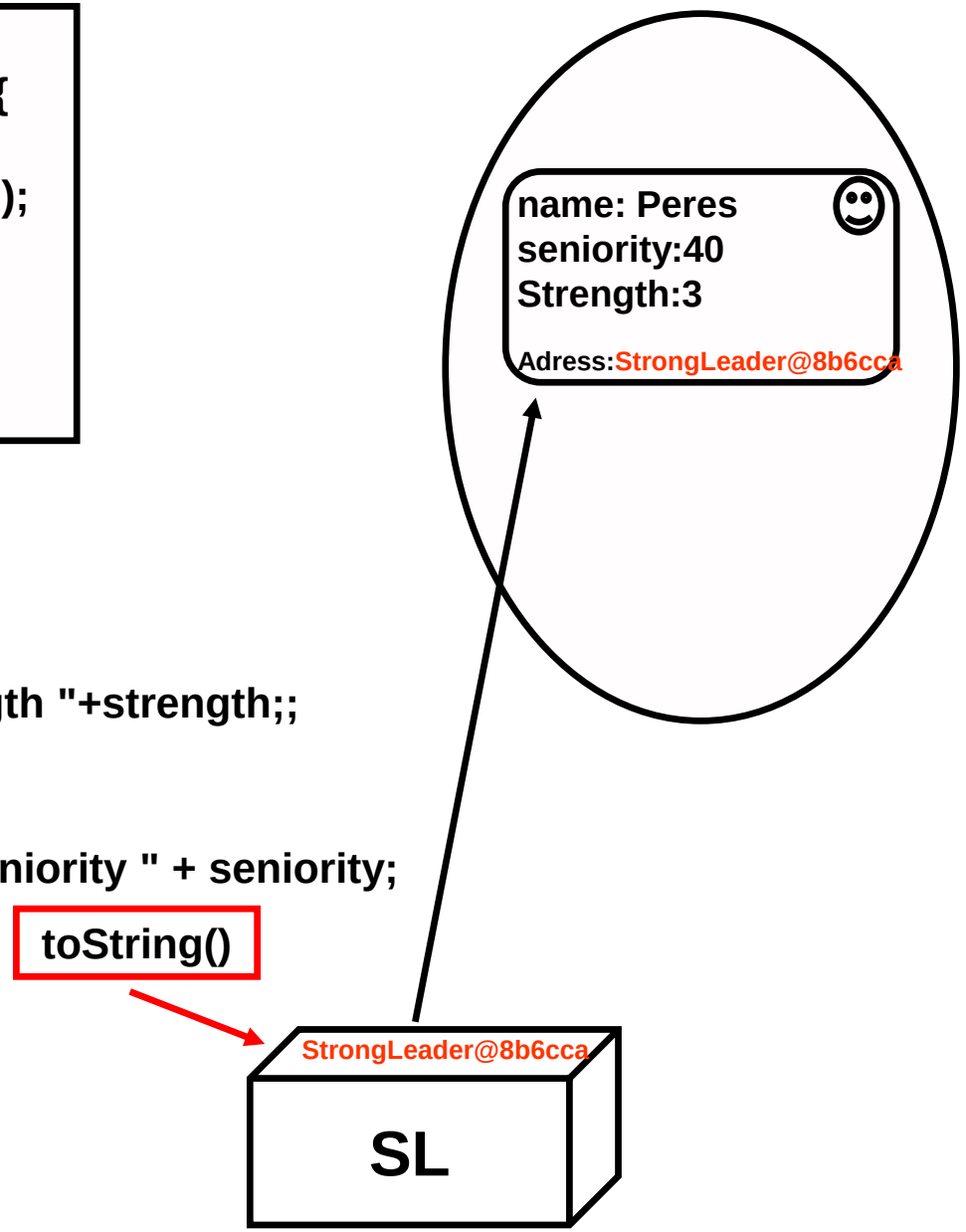
```
public class Tester {
    public static void main(String[ ] args){
        StrongLeader SL;
        SL=new StrongLeader("Peres",40,3);
        SL.whoAml();
        System.out.println(SL);
    }
}
```

```
public String toString(){
    return super.toString()+" and stringth "+strength;;
}
```



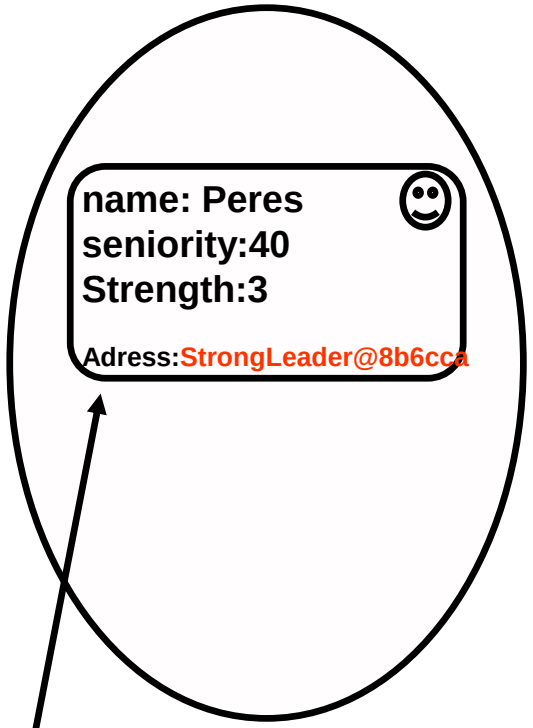
```
public class Tester {
    public static void main(String[ ] args){
        StrongLeader SL;
        SL=new StrongLeader("Peres",40,3);
        SL.whoAml();
        System.out.println(SL);
    }
}
```

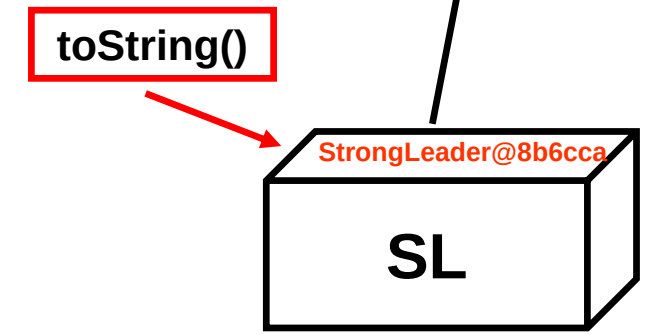
```
public String toString(){
        return super.toString()+" and stringth "+strength;;
}
public String toString(){
        return "Leader " + name + " with seniority " + seniority;
}
```



```
public class Tester {
    public static void main(String[ ] args){
        StrongLeader SL;
        SL=new StrongLeader("Peres",40,3);
        SL.whoAml();
        System.out.println(SL);
    }
}
```

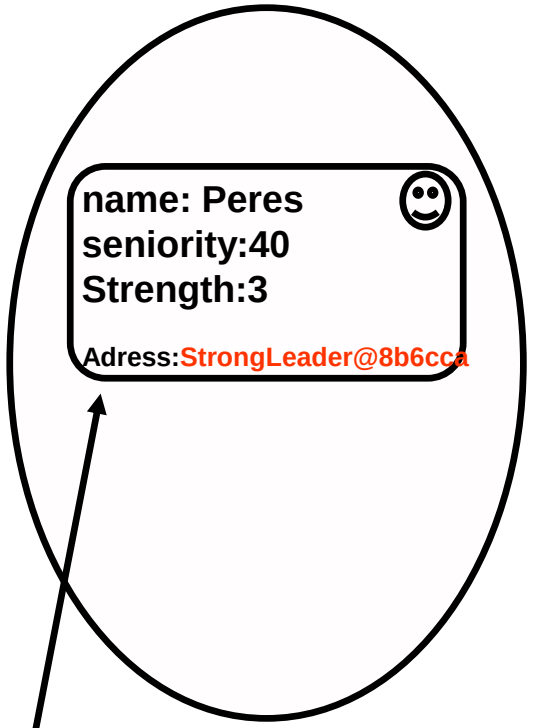
```
public String toString(){
        return super.toString()+" and stringth "+strength;;
}
public String toString(){
        return "Leader " + name + " with seniority " + seniority;
}
```

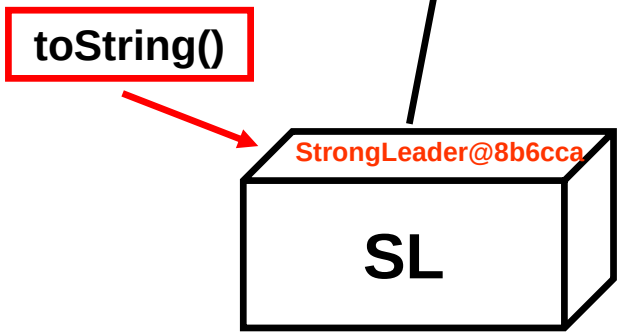




```
public class Tester {
    public static void main(String[ ] args){
        StrongLeader SL;
        SL=new StrongLeader("Peres",40,3);
        SL.whoAml();
        System.out.println(SL);
    }
}
```

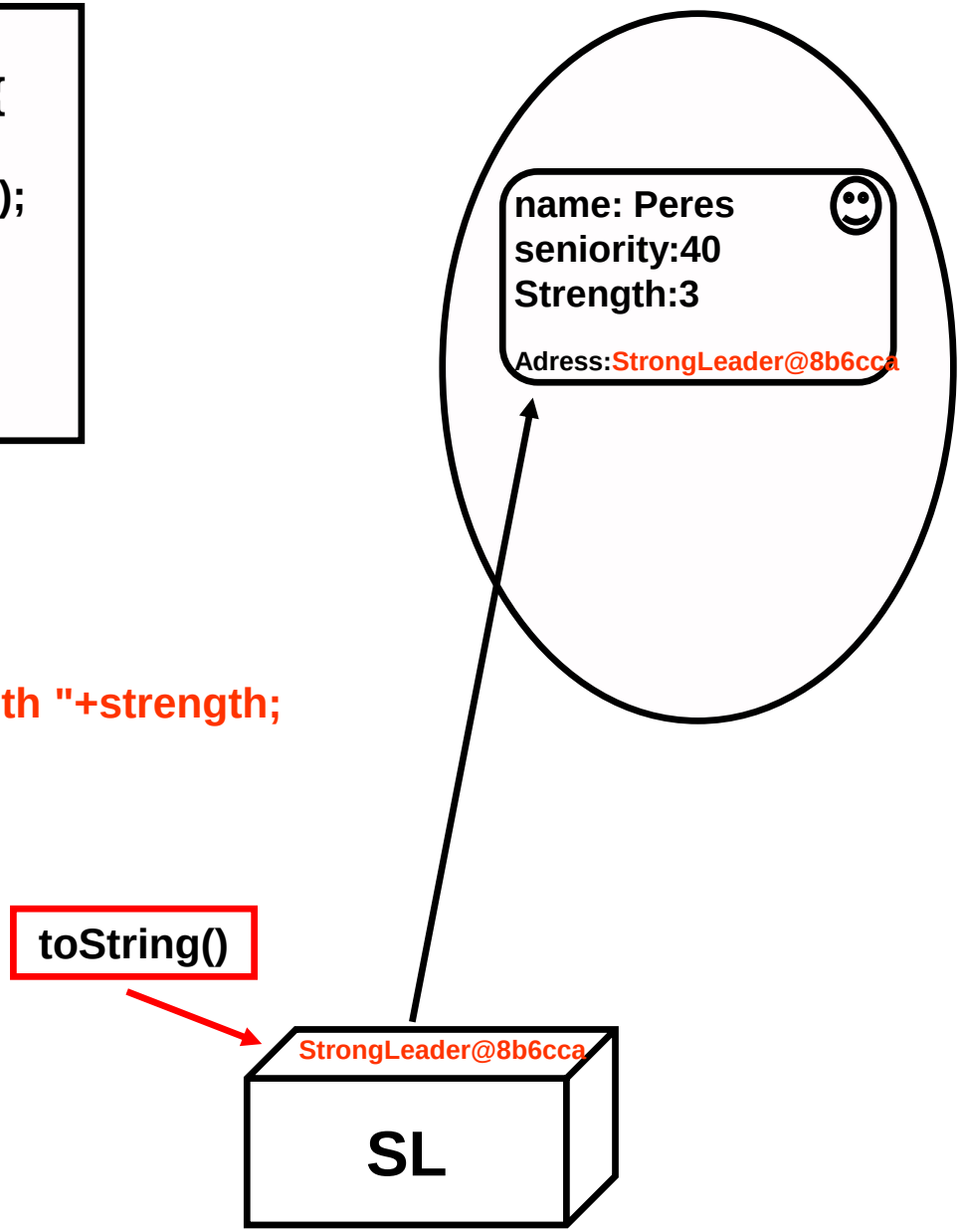
```
public String toString(){
        return super.toString()+" and stringth "+strength;;
}
public String toString(){
        return "Leader " + name + " with seniority " + seniority;
}
```





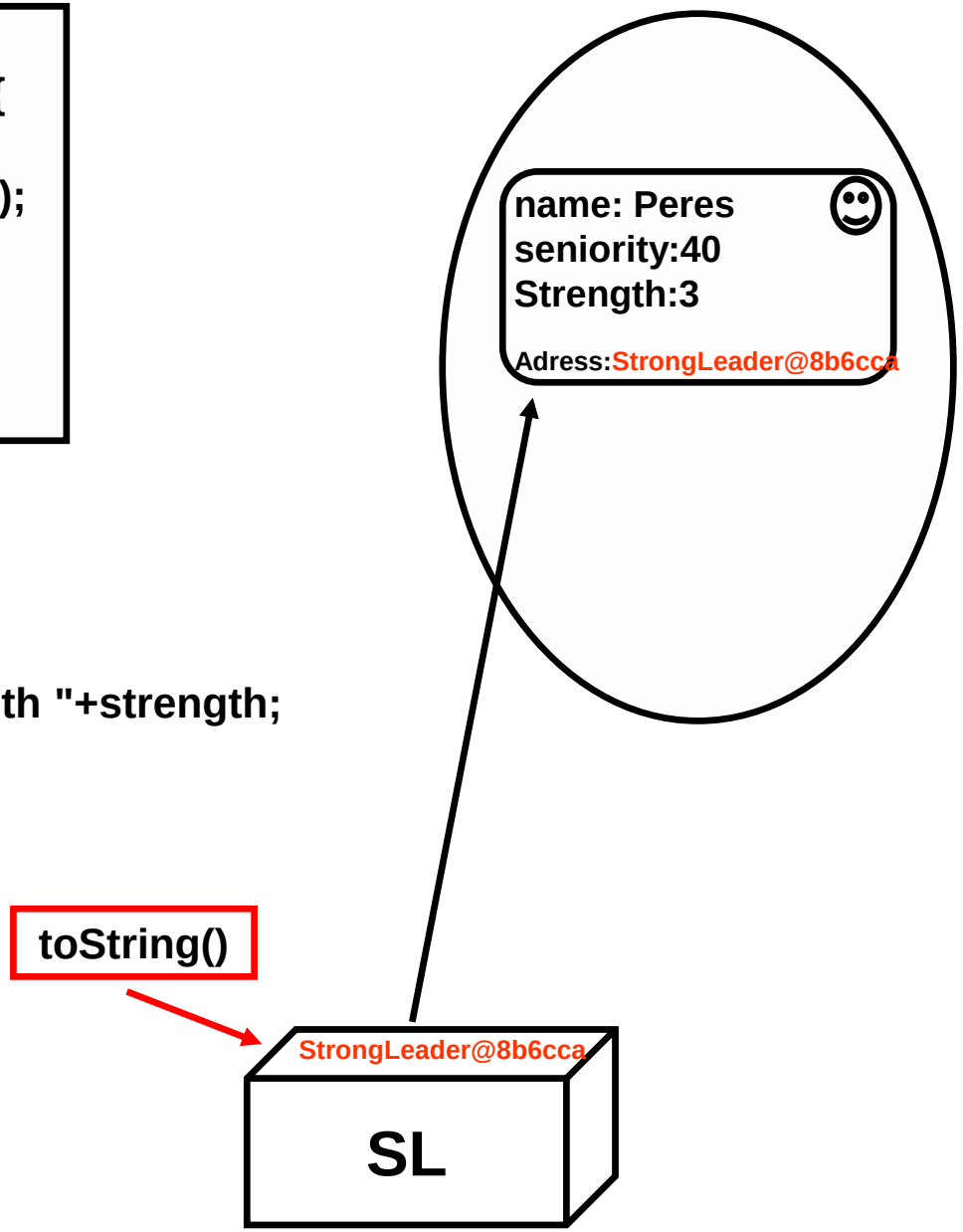
```
public class Tester {
    public static void main(String[ ] args){
        StrongLeader SL;
        SL=new StrongLeader("Peres",40,3);
        SL.whoAml();
        System.out.println(SL);
    }
}
```

```
public String toString(){
    return super.toString()+" and stringth "+strength;
}
```

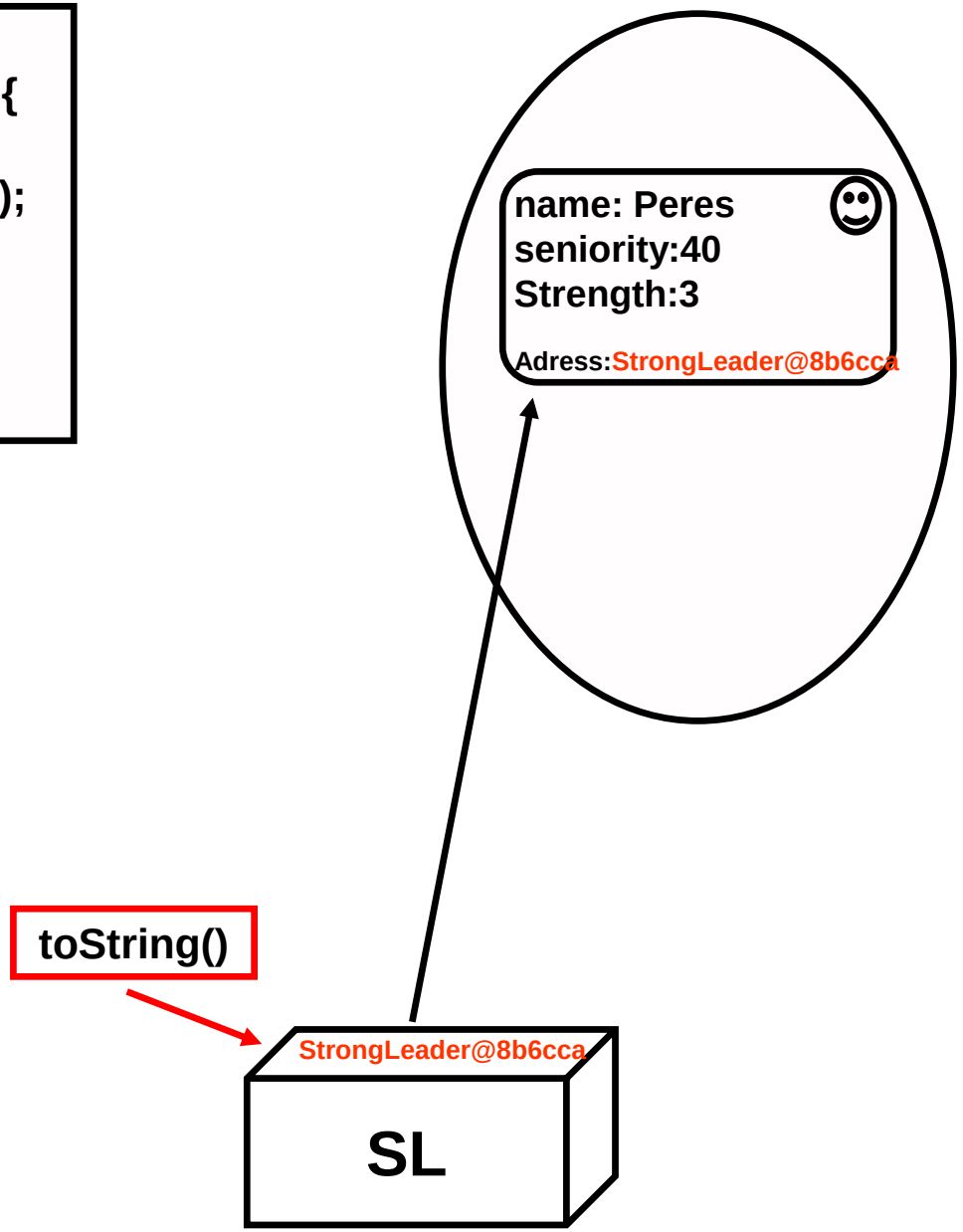


```
public class Tester {
    public static void main(String[ ] args){
        StrongLeader SL;
        SL=new StrongLeader("Peres",40,3);
        SL.whoAml();
        System.out.println(SL);
    }
}
```

```
public String toString(){
        return super.toString()+" and stringth "+strength;
}
```



```
public class Tester {
    public static void main(String[ ] args) {
        StrongLeader SL;
        SL=new StrongLeader("Peres",40,3);
        SL.whoAml();
        System.out.println(SL);
    }
}
```



R
E
M
E
M
B
E
R

- Java guarantees that a class constructor is called whenever an instance of that class is created.
- Java also guarantees that the constructor is called whenever an instance of any ***subclass*** is created.

```
public class B extends A {
    private int y;
    public B(){
      //super()   → x=10
      y=0;
    }
    public B(int x,int y){
      //super() →x=10
      this.y=y; }
}
```

```
public class A {
    protected int x;
    public A(){
      x=10;
    }
    public A(int x){
      this.x=x;
    }
}
```

הבנאי הריק של מחלקת הבסיס מופעל באופן אוטומטי מכל מחלקה נגזרת ובתנאי שאותו בנאי ריק קיים במחלקת הבסיס אחרת מתקבלת שגיאת קומפילציה

```
public class B extends A {
    private int y;
    public B(){
     //super() → x=10
      y=0;
    }
    public B(int x,int y){
      super(x);
      this.y=y;
    }
}
```

Remember:- every constructor method must call (explicitly or implicitly) its superclass constructor method

```
public class A {
    protected int x;

    public A(int x){
        this.x=x;
    }
}
```

```
public class B extends A {
    private int y;
    public B(){
      //super()!!!
       y=0;
    }
    public B(int x,int y){
        super(x);
        this.y=y;
    }
}
```

Error: B() calls implicitly A() but A() not found!!!

Constructor Overloading

- ריבוי בנאים - ניתן להגדיר יותר מבנאי אחד
- כל בנאי שונה מהאחר ע"י **רשימת הפרמטרים** שהוא מקבל
- זיהוי הבנאי מתבצע בשלב הקמת האובייקט מיד אחרי new
- איך עושים זאת?
- הבא ונסתכל שוב במחלקה Person

```
public class Person {
    private String firstName;
    private String secondName;
    private int id;
    public Person(){
      firstName="?";
      secondName="?";
      id=0;
    }
    public Person(int id) {
      this.id=id;
    }
    public Person(String name ,String family ,int id) {
      firstName=name;
      secondName=family;
      this.id=id;
    }
    public void whoAml(){…}
}
```

Constructor overload

זיהוי הבנאי המתאים מתבצעת ע"י בדיקת רשימת הפרמטרים שנשלחה לבנאי

```
public class A {
    private int x , y;
    public A(){
        x=0;
        y=0;
    }
    public A(int x){
        this.x=x;
        y=0;
    }
    public A(int x , int y){
        this.x=x;
        this.y=y;
    }
     public String toString(){
        return "x="+x+" y="+y;
     }
}
```

```
public class Tester {
    public static void main(String[ ] args){
        A obj1=new A();
        A obj2=new A(2);
        A obj3=new A(3,4);
        System.out.println(obj1);//x=0 y=0
        System.out.println(obj2);//x=2 y=0
        System.out.println(obj3);//x=3 y=4
    }
}
```

Method Overloading

- זיהוי השיטה מתבצע ע"י השם + רשימת הפרמטרים שמקבלת
- ניתן להגדיר יותר משיטה אחת בעלות אותו שם
  - ✓ כל השיטות בעלות אותו שם
  - ✓כל שיטה שונה מהאחרת ע"י **רשימת הפרמטרים** שהיא מקבלת
- איך עושים זאת?
- הבא ונסתכל בדוגמה בעמוד הבא

Methods Overloading

הוספת השיטה
set

```
public class A {
 private int x , y;
 public A(){…}
 public A(int x){…}
 public A(int x , int y){…}
 public void set(){
      x=y=0;
 }
 public void set(int x){
    this.x=x;
 }
 public void set(int x , int y){
    this.x=x;
    this.y=y;
 }
 public String toString(){..}
}
```

```
public class Tester {
   public static void main(String[] args){
      A obj1=new A();
      A obj2=new A(2);
      A obj3=new A(3,4);
      System.out.println(obj1); //x=0 y=0
      System.out.println(obj2); //x=2 y=0
      System.out.println(obj3); //x=3 y=4
      obj3.set();
      System.out.println(obj3); //x=0 y=0
      obj3.set(5);
      System.out.println(obj3); //x=5 y=0
      obj3.set(6,7);
      System.out.println(obj3); //x=6 y=7
   }
}
```

```java
public class A {
    private int x;
    private double y;
    public A(){
      x=0;
      y=0;
    }
    public A(int x){
       this.x=x;
    }
    public A(double y){
      this.y=y;
    }
    public A(int x , double y){
       this.x=x;
       this.y=y;
    }
    public String toString(){
      return "x="+x+" y="+y;
    }
}
```

```java
public class Tester {
  public static void main(String[] args){
    A obj1=new A();
    A obj2=new A(2);
    A obj3=new A(2.5);
    A obj4=new A(3,4.0);
    A obj5=new A(3,4);
    //A obj6=new A(3.0,4); //Error
    System.out.println(obj1); //x=0 y=0.0
    System.out.println(obj2); //x=2 y=0.0
    System.out.println(obj3); //x=0 y=2.5
    System.out.println(obj4); //x=3 y=4.0
    System.out.println(obj5); //x=3 y=4.0
  }
}
```

Methods Overloading

```
public class A {
    private int x; private double y;
    public A() {…}
    public A(int x) {…}
    public A(double y) {…}
    public A(int x ,double y) {…}
    public void set(){
        x=0; y=0;
    }
    public void set(int x){
        this.x=x;
        y=0;
    }
   public void set(double y){
        this.x=0;
        y=this.y;
    }
    public void set(int x , double y){
        this.x=x;
        this.y=y;
    }
    public String toString(){…}
}
```

הוספת השיטה set

```
public class Tester {
    public static void main(String[] args){
        A obj1=new A();
        A obj2=new A(2);
        A obj3=new A(2.5);
        A obj4=new A(3,4.5);
        System.out.println(obj1); //x=0 y=0.0
        obj1.set();
        System.out.println(obj1); //x=0 y=0.0
        obj1.set(5);
        System.out.println(obj1); //x=5 y=0.0
        obj1.set(1.5);
        System.out.println(ob1); //x=0 y=1.5
        obj1.set(1,1.5);
        System.out.println(ob1); //x=1 y=1.5
    }
}
```

```
public class A {
    private int x;
    private double y;
    public A( ) {  x=0;    y=0;  }
    public A(int x) {   this.x=x;  }
    public A(double y) {  this.y=y; }
    public A(int x , double y) {
        this.x=x;     this.y=y;
    }
    public A(double y , int x) {
        this.x=x;    this.y=y;
    }
    public void set(){
        x=0; y=0;
    }
    public void set(int x ) {  this.x=x; }
    public void set(int x , double y) {
        this.x=x;  this.y=y;
    }
    public void set(double y , int x) {
        this.x=x;  this.y=y;
    }
    public String toString( ) {
       return "x=" + x + " y=" + y;
    }
}
```

constructor Overloading

method Overloading

שימו לב:
לסדר של הפרמטרים יש חשיבות.
שיטות (גם בנאים) שמקבלות פרמטרים
זהים אבל עם סדר שונה הן שיטות שונות.

```
public class Tester {
    public static void main(String[ ] args){

      A obj1=new A();
      A obj2=new A(2);
      A obj3=new A(2.5);
      A obj4=new A(3 , 4.0); //x=3 , y=4.0
      A obj5=new A(5.5 , 1); //x=1 , y=5.5

      obj1.set(1 , 2.5); // x=1 , y=2.5
      obj1.set(1.7 , 2); // x=2 , y=1.7    }
}
```

```
public class A {
    private int x;
    public A(){
      x=0;
    }
    public A(int x){
       this.x=x;
    }
    public void set(int x){
       this.x=x;
    }
    public int set(int x){
       this.x=x;
      return x;
    }
    public String toString(){
      return "x="+x;
    }
}
```

שימו לב:
הגדרת שיטות עם שמות זהים ואשר מקבלות פרמטרים מטיפוסים זהים (באותו הסדר) גורר לשגיאת קומפילציה אפילו אם הערך המוחזר ע"י השיטות שונה .

Error
method set(int) is already defined in class A

```
public class Tester {
    public static void main(String[ ] args){

      A obj1=new A();

      obj1.set(1); // Error
   }
}
```

ה compiler מבולבל ולא יודע מה לעשות יש לו שתי שיטות זהות בשם שלהן וברשימת הפרמטרים שהן מקבלות

```
public class A {
    private int x;
    private double y;

   public A(int x , double y) {
        this.x=x;     this.y=y;
    }
    public A(double y , int x) {
        this.x=x;    this.y=y;
    }

   public void set(int x , double y) {
        this.x=x;  this.y=y;
    }
    public void set(double y , int x) {
        this.x=x;  this.y=y;
    }

    public String toString( ) {
      return "x=" + x + " y=" + y;
    }
}
```

```
public class Tester {
    public static void main(String[ ] args){
      A obj1=new A(3 , 3.5);

      A obj2=new A(3 , 3);

      obj1.set(3 , 3);
    }
}
```

Error

Error:

- Reference to A is ambiguous, both constructors A(int,double) in A and constructor A(double,int) in A match
- Reference to set is ambiguous, both methods set(int,double) in A and method set(double,int) in A match

- השגיאה היא לא בגלל הבנאים והשיטות אלא בגלל הפניה אשר יוצרת בלבול למהדר. וזאת בגלל ה casting שיכול להתאים לשני הבנאים ולשתי השיטות.

- שאלה במידה ומשמטים אחד מהבנאים וכן אחת מהשיטות האם עדיין נקבל שגיאה? תשובה לא

- שאלה איך ניתן להתגבר על הבעיה מבלי להשמיט בנאי או שיטה?
תשובה 1: הוספת בנאי\שיטה מתאימה (עמוד הבא)
תשובה 2: פניה ישירה לבנאי\שיטה המתאימים .

```
public class A {
    private int x;
    private double y;

   public A(int x , double y) {
        this.x=x;     this.y=y;
    }
    public A(double y , int x) {
        this.x=x;    this.y=y;
    }
   public A(int y , int x) {
        this.x=x;    this.y=y;
    }
   public void set(int x , double y) {
        this.x=x;  this.y=y;
    }
    public void set(double y , int x) {
        this.x=x;  this.y=y;
    }
    public void set(int y , int x) {
        this.x=x;  this.y=y;
    }
    public String toString( ) {
      return "x=" + x + " y=" + y;
    }
}
```

```
public class Tester {
    public static void main(String[ ] args){
      A obj1=new A(3 , 3.5);

      A obj2=new A(3 , 3);

      obj1.set(3 , 3);
    }
}
```

עכשיו זה טוב
פניה ישירה לשיטה\בנאי המתאימים

Methods Overriding

- **A child's method overrides a parent's method when it has the same signature as a parent method. (for example toString() of class Leader overrides toString() of Person)**

- **Once overriding occurs - the parent has its method, and the child has its own method with the same signature.**

- **Remember that the signature of a method is the name of the method and its parameter list.**

- **Sometimes you want a child class to have its own method, but that method includes everything the parent's method does. You can use the super reference in this situation.(for example toString() In class StrongLeader call super.toString() in Leader;**

דוגמה

sum הוגדרה במחלקה A
אבל לא הוגדרה במחלקה B

```
public class A {
  protected int x , y;

  public A(int x , int y){
     this.x=x;
     this.y=y;
  }
  public void sum(){
    System.out.println("A: Sum="+ (x+y));
  }
}
```

```
public class B extends A {
   private int z;
   public B(int x , int y , int z) {
        super ( x , y );
        this.z = z;
   }
}
```

```
public class Tester {
  public static void main(String[ ] args){

    A a=new A(1,2);
    a.sum();
    // A: Sum=3

    B b=new B(1,2,3);
    b.sum();
    // A: Sum=3

    }
}
```

B מקבלת בירושה את ה-sum אשר הוגדרה ב A

```
public class A {
  protected int x , y;

  public A(int x , int y){
    this.x=x;
    this.y=y;
  }
  public void sum(){
    System.out.println("A: Sum="+ (x+y));
  }
}
```

```
public class B extends A{
  private int z;

  public B(int x,int y,int z){
    super(x,y);
    this.z=z;
  }
 @Override
  public void sum(){
    System.out.println("B: Sum="+ (x+y+z));
  }
}
```

sum הוגדרה במחלקה A
B מקבלת בירושה את ה-sum של A
אבל sum הוגדרה מחדש ב B ולכן
דורסת את זו שהתקבלה מ-A

```
public class Tester {
   public static void main(String[ ] args){

     A a=new A(1,2);
     a.sum();  //A: Sum=3

     B b=new B(1,2,3);
     b.sum();  // B: Sum=6
   }
}
```

sum הוגדרה במחלקה A
B מקבלת בירושה את ה-sum של A
sum הוגדרה מחדש ב B ולכן דורסת את זו
שהתקבלה מ-A אבל מתוך ה-sum שב-B יש
פניה ישירה ל-sum של A ע"י שימוש ב
super

```
public class A {
    protected int x , y;
    public A(int x , int y) {
        this.x=x;
        this.y=y;
    }
    public int sum() {
        return x + y;
    }
}
```

```
public class B extends A {
    private int z;
    public B(int x ,int y , int z) {
        super( x , y);
        this.z = z;
    }
    @Override
    public int sum() {
        return super.sum() + z;
    }
}
```

```
public class Tester {
    public static void main(String[ ] args) {

        A a=new A(1,2);
        System.out.println( a.sum() );  //3

        B b=new B(1,2,3);
       System.out.println( b.sum() ); //6
    }
}
```

```
public class A {
    protected int x , y;
    public A(int x , int y) {
        this.x=x;
        this.y=y;
    }
    public int sum() {
        return x + y;
    }
}
```

שימו לב
Sum הוגדרה מחדש ב B אבל עם שינוי קל
השיטה sum ב B לא מחזירה ערך בניגוד
לשיטה sum ש ב A . זה גורר לשגיאת
קומפילציה (למה?).

```
public class Tester {
    public static void main(String[ ] args) {

        B b=new B(1,2,3);
        b.sum() ;
    }
}
```

```
public class B extends A {
    private int z;
    public B(int x ,int y , int z) {
        super( x , y);
        this.z = z;
    }
    @Override
    public void sum() {
        System.out.println( super.sum() + z);
    }
}
```

Compilation Error
sum() in B cannot override sum() in A
return type void is not compatible with int

ה compiler מזהה שתי שיטות הזהות בשם
וברשימת הפרמטרים שהן מקבלות לכן הוא
מצפה לבצע override אבל הוא מגלה
שהחתימות המלאות (כולל הערך המוחזר) של
הפונקציות לא ממש זהה ולכן הוא צועק

שימו לב
אין כאן דריסה (overriding) אלא
העמסה (overloading).

```
public class A {
    protected int x , y;
    public A(int x , int y) {
        this.x=x;
        this.y=y;
    }

    public int sum(int x) {
        return x + y;
    }
}
```

```
public class Tester {
    public static void main(String[ ] args) {

        B b=new B(1,2,3);
        b.sum() ;
    }
}
```

```
public class B extends A {
    private int z;
    public B(int x ,int y , int z) {
        super( x , y);
        this.z = z;
    }

    public void sum() {
        System.out.println( “Sum:” + super.sum(2) + z);
    }
}
```

Sum: 43

```java
public class A {
    protected int x;
    public A(){
        x=0;
    }
    public A(int x){
        this.x=x;
    }
    public void set(int x){
        this.x=x;
    }
    public String toString(){
        return "x="+x;
    }
}
```

```java
public class B extends A {
    private int y;
    public B(){
        // automaticly calls super()
        y=0;
    }
    public B(int x,int y){
        super(x);
        this.y=y;
    }
    public void set(int y){
        this.y=y;
    }
    public String toString(){
        return super.toString()+" y="+y;
    }
}
```

```java
public class Tester {
    public static void main(){
        A a = new A();
        a.set(1);
        System.out.println(a); //x=1

        B b = new B();
        b.set(2);
        System.out.println(b); //x=0 , y=2

    }
}
```

set הוגדרה מחדש ב B ולכן היא דורסת את זו של A
שאלה איך ניתן לפנות ל set של A מ B?
תשובה: הבא ונעבור לעמוד הבא

```java
public class A {
    protected int x;
    public A(){
        x=0;
    }
    public A(int x){
        this.x=x;
    }
    public void set(int x){
        this.x=x;
    }
    public String toString(){
        return "x="+x;
    }
}
```

```java
public class B extends A {
    private int y;
    public B(){
        // automaticly calls super()
        y=0;
    }
    public B(int x,int y){
        super(x);
        this.y=y;
    }
    public void set(int x){
        super.set(x); // super.x=x; or this.x=x;
    }
    public String toString(){
        return super.toString()+" y="+y;
    }
}
```

```java
public class Tester {
    public static void main(){
        A a = new A();
        a.set(1);
        System.out.println(a); //x=1

        B b = new B();
        b.set(2);
        System.out.println(b); //x=2 , y=0

    }
}
```

ניתן להשתמש ב super כדי לפנות ישירות למשתנה או שיטה במחלקת הבסיס.

```java
public class A {
    protected int x;
    public A(){
        x=0;
    }
    public A(int x){
        this.x=x;
    }
    public void set(){
        x=0;
    }
    public void set(int x){
        this.x=x;
    }
    public String toString(){
        return "x="+x;
    }
}
```

```java
public class B extends A {
    private int y;
    public B(){
      // automaticly calls super()
      y=0;
    }
    public B(int x,int y){
        super(x);
        this.y=y;
    }
    public void set(int x , int y){
        super.set(x);
        this.y=y;
    }
    public String toString(){
        return super.toString()+" y="+y;
    }
}
```

```java
public class Tester {
    public static void main(){
        A a = new A();
        a.set(1);
        System.out.println(a); //x=1

        B b = new B();
        b.set(1,2);
        System.out.println(b); //x=1 , y=2

    }
}
```

this is not overriding
It is overloading

END