Η Κλάση & Δημιουργία Αντικειμένων

Γιώργος Θάνος
gthanos@uth.gr
Γραφείο: Γ5/8, 3ος όροφος
Γκλαβάνη 37

Η κλάση μέσα από ένα παράδειγμα

```
public class Rectangle {
   public int width;
   public int height;

   public Rectangle(int initWidth, int initHeight) {
     width = initWidth;
     height = initHeight;
   }

   void setWidth(int newWidth ) { width = newWidth; }
   void setHeight(int newHeight ) { height = newHeight; }
   int getWidth() { return width;  }
   int getHeight() { return height;  }

   public int area() { return width * height; }

 }
```

Ορίζουμε την κλάση Rectangle που απεικονίζει ένα ορθογώνιο παραλληλόγραμμο. Η κλάση περιέχει

α) τα πεδία **width** και **height**

β) τις μεθόδους

public void setWidth(int newWidth);
public void setHeight(int newHeight);
public int getWidth();
public int getHeight();
public int area();

γ) τον κατασκευαστή

public Rectangle(int setWidth, int setHeight);

Τα πεδία της κλάσης

Κανόνες ονοματολογίας

Τα ονόματα των μεταβλητών είναι case-sensitive. Κάθε μεταβλητή μπορεί να ξεκινά με ένα γράμμα της αλφαβήτου (κεφαλαίο ή μικρό), τον χαρακτήρα '$' ή τον χαρακτήρα '_'.

Οι υπόλοιποι χαρακτήρες των μεταβλητών μπορεί να είναι γράμματα (κεφαλαία και μικρά), αριθμοί και οι χαρακτήρες '$', '_'. Αποφύγετε στην ονοματολογία σας μεταβλητές που έχουν μόνο ένα γράμμα (π.χ. int s, double c) εκτός και εάν πρόκειται για μεταβλητές δείκτες που χρησιμοποιούνται σε επανάληψη (π. χ for, while).

Αν το όνομα της μεταβλητής σας αποτελείται από μία λέξη τότε γράψτε την λέξη αυτή με μικρά γράμματα (π.χ. role, colour, speed). Εάν η μεταβλητή σας περιέχει δύο λέξεις ενώστε τις λέξεις αυτές κάνοντας το πρώτο γράμμα κάθε λέξης κεφαλαίο (π.χ. adminRole, brighterColour, maxSpeed, maxSpeedTest).

Εάν η μεταβλητή σας ορίζεται ως σταθερά, δηλαδή χαρακτηρίζεται από το πρόθεμα static final (π.χ. static final double PI = 3.14159265359;), τότε χρησιμοποιήστε εξ ολοκλήρου κεφαλαία αντί για μικρά γράμματα.

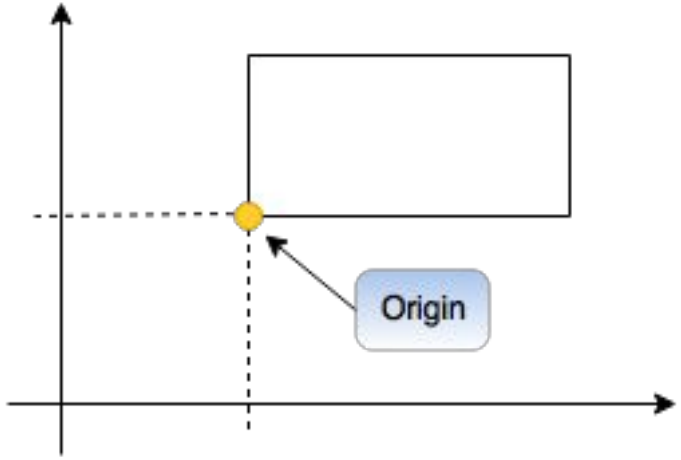
Τύποι δεδομένων των πεδίων της κλάσης

- **primitive (βασικούς) τύπους δεδομένων:** Παραδείγματα βασικών τύπων είναι i**nt, long, char, double, floa**t κ.α.

- **reference (αναφορικούς) τύπους δεδομένων:** τύπους δεδομένων οι οποίοι περιγράφονται από κλάσεις

Παράδειγμα Αναφορικών τύπων δεδομένων

- Ας υποθέσουμε ότι στο παράδειγμα του ορθογωνίου παραλληλογράμμου θέλουμε να προσθέσουμε την θέση του παραλληλογράμμου στον χώρο.
- Η θέση του παραλληλογράμμου μπορεί να προσδιοριστεί από την κλάση **Point** η οποία περιγράφει ένα σημείο στο χώρο. Θεωρήστε ότι το σημείο προσδιορίζει την κάτω αριστερή γωνία του παραλληλογράμμου στον χώρο, όπως φαίνεται στο διπλανό σχήμα.
- Η κλάση **Rectangle** μετασχηματίζεται ώστε να χρησιμοποιεί ένα πεδίο τύπου Point.



Οι μέθοδοι της κλάσης

Οι μέθοδοι της κλάσης έχουν απευθείας πρόσβαση στα πεδία της κλάσης

Ορισμός μεθόδων της κλάσης

Ο ορισμός οποιασδήποτε μεθόδου στη Java περιέχει τα εξής πεδία

```
<πρ. πρόσβασης> <επιστ. τύπος> <όνομα> (<λίστα περαμέτρων>) {

}
```

1. Προσδιοριστές πρόσβασης, όπως **`public`** ή **`private`**.
2. Τον τύπο της επιστρεφόμενης τιμής ή void εάν η μέθοδος δεν επιστρέφει κάτι.
3. Το όνομα της μεθόδου.
4. Η λίστα των παραμέτρων μέσα σε παρένθεση, χωρισμένες με κόμμα. Κάθε παράμετρος εκτός από το όνομά της οφείλει να διαθέτει και τον τύπο της, π.χ. int speed ή double latitude. Εάν δεν υπάρχουν παράμετροι το περιεχόμενο των παρενθέσεων είναι κενό.
5. Το κυρίως σώμα της μεθόδου, το οποίο περιέχεται μέσα σε αγκύλες {}

Κανόνες ονοματολογίας

1. Ισχύει και εδώ ο “κανόνας” ότι τα ονόματα ξεκινούν με μικρό γράμμα και εάν πρόκειται για σύνθετα ονόματα με περισσότερες από μία λέξεις αυτές χωρίζονται με κεφαλαία γράμματα. Η πρώτη λέξη κάθε μεθόδου συνηθίζεται να είναι ρήμα. Μερικά παραδείγματα:
   - `run`
   - `runFast`
   - `getBackground`
   - `getFinalData`
   - `compareTo`
   - `setLatitude`
   - `isEmpty`

Παραδείγματα Μεθόδων

```
public class Rectangle {
   public int width;
   public int height;

   public Rectangle(int initWidth, int initHeight) {
     width = initWidth;
     height = initHeight;
   }

   public void setWidth(int newWidth ) { width = newWidth; }
   public void setHeight(int newHeight ) { height = newHeight; }
   public int getWidth() { return width;  }
   public int getHeight() { return height;  }
   public void resize(int dw, int dh) {

     width += dw; height += dh;

   }

  public int getArea() { return width * height; }
}
```

Υπερφόρτωση (overloading) των μεθόδων μίας κλάσης

Χαρακτηριστικό του Αντικειμενοστραφούς Προγραμματισμού είναι η **υπερφόρτωση** των μεθόδων της κλάσης.

Με τον όρο **υπερφόρτωση** εννοούμε την δυνατότητα που μας δίνει ο Αντικειμενοστραφής Προγραμματισμός να ορίσουμε σε μία κλάση δύο μεθόδους με το ίδιο όνομα αλλά διαφορετικά ορίσματα. Τα ορίσματα μπορούν να διαφέρουν **a)** ως προς τον τύπο τους **b)** ως προς τον αριθμό τους και **c)** ως προς την σειρά τους.

Αν και ο compiler της Java αντιλαμβάνεται την διαφορά, δεν αποτελεί ορθή προγραμματιστική πρακτική να έχετε μεθόδους με τον ίδιο αριθμό και τύπο ορισμάτων, τα οποία όμως είναι διατεταγμένα με διαφορετική σειρά.

Υπερφόρτωση μεθόδων - Παράδειγμα

Ας υποθέσουμε ότι έχουμε την κλάση DrawChars η οποία αποτελεί μέρος ενός υποθετικού προγράμματος εκτύπωσης χαρακτήρων στην οθόνη του υπολογιστή. Παρουσιάζουμε την μέθοδο **draw** η οποία διαθέτει διαφορετικές εκδόσεις ανάλογα με τι εκτυπώνεται κάθε φορά.

Το ποια έκδοση χρησιμοποιείται σε κάθε κλήση της μεθόδου αποφασίζεται από τον compiler με βάση το είδος και τον αριθμό των ορισμάτων

```
public class DrawChars {
    ...
    public void draw(String s) {
        ...
    }
    public void draw(int i) {
        ...
    }
    public void draw(double f) {
        ...
    }
    public void draw(int i, double f) {
        ...
    }
}
```

Οι κατασκευαστές της κλάσης

Οι κατασκευαστές μιας κλάσης είναι **ειδικές μέθοδοι** με τα εξής χαρακτηριστικά:

1. έχουν ΠΑΝΤΑ προσδιοριστή πρόσβασης **public**.
2. δεν έχουν επιστρεφόμενο τύπο, και
3. το όνομα τους ταυτίζεται με το όνομα της κλάσης.

Παράδειγμα

```
public class Rectangle {
  int width;
  int height;
  Point origin;

  public Rectangle(int initWidth, int initHeight, Point initOrigin) {
    width = initWidth;
    height = initHeight;
    origin = initOrigin;
  }
}
```

Ο ρόλος του κατασκευαστή

Δεσμεύει την απαραίτητη μνήμη για την δημιουργία του αντικειμένου και αρχικοποιεί τα πεδία της κλάσης με βάση τις αρχικές παραμέτρους με τις οποίες καλείται.

Η κλήση του κατασκευαστή γίνεται πάντα με χρήση του τελεστή **new**, όπως θα δούμε στη συνέχεια.

Default κατασκευαστής

Κάθε κλάση έχει τουλάχιστον ένα κατασκευαστή. Εάν δεν δηλώσετε κατασκευαστή σε κάποια κλάση ο compiler δημιουργεί τον *default* κατασκευαστή, δηλ. ένα κατασκευαστή χωρίς ορίσματα και αρχικοποιεί τα πεδία της κλάση σε **μηδέν (0)** για βασικούς τύπους ή σε ***null*** για αναφορικούς/σύνθετους τύπους.

```
public class Rectangle {
  int width;
  int height;
  Point origin;

  public Rectangle() {
  }
}
```

Στο προηγούμενο παράδειγμα της κλάσης **Rectangle** εάν υποθέσουμε ότι αυτή δεν έχει κατασκευαστή, ο *compiler* θα δημιουργήσει τον διπλανό κατασκευαστή.

ο οποίος αρχικοποιεί τις τιμές ως εξής

```
width = 0;
height = 0;
origin = null;
```

Πολλαπλοί κατασκευαστές σε μία κλάση

Όπως μπορούμε να έχουμε πολλές μεθόδους με το ίδιο όνομα αλλά διαφορετικά ορίσματα, έτσι μπορούμε να έχουμε και πολλούς κατασκευαστές με διαφορετικά ορίσματα.

Δείτε το παράδειγμα για την κλάση **Rectangle** της προηγούμενης ενότητας. Στο παράδειγμα αυτό η κλάση διαθέτει τους παρακάτω τρεις κατασκευαστές

```
public Rectangle(int initWidth, int initHeight, Point initOrigin) {
   width = initWidth;
   height = initHeight;
   origin = initOrigin;
 }

public Rectangle(int initWidth, int initHeight, int xPos, int yPos) {
   width = initWidth;
   height = initHeight;
   origin = new Point(xPos, yPos);
 }

public Rectangle(int initWidth, int initHeight) {
   width = initWidth;
   height = initHeight;
   origin = null;
 }
```

Δημιουργία Αντικειμένων

Δημιουργία Αντικειμένων

Η δημιουργία αντικειμένων γίνεται με χρήση του τελεστή **new**. Για παράδειγμα, για να δημιουργήσουμε ένα αντικείμενο της κλάσης **Point** αρκεί να γράψουμε

```
Point p = new Point(3,5); // δημιουργεί ένα αντικείμενο με συντεταγμένες 3,5
```

Παράδειγμα δημιουργίας αντικειμένων

Στο επόμενο παράδειγμα επαναορίζονται οι κλάσεις **Point** και **Rectangle** και προστίθεται η κλάση **CreateObjectDemo** που περιέχει τη στατική μέθοδο **main** μέσα στην οποία δημιουργούνται τα αντικείμενα.

```
public class CreateObjectDemo {

    public static void main(String[] args) {

        // Declare and create a point object and two rectangle objects.
        Point originOne = new Point(23, 94);
        Rectangle rectOne = new Rectangle(100, 200, originOne);
        Rectangle rectTwo = new Rectangle(50, 100);

        // display rectOne's width, height, and area
        System.out.println("Width of rectOne: " + rectOne.getWidth() );
        System.out.println("Height of rectOne: " + rectOne.getHeight() );
        System.out.println("Area of rectOne: " + rectOne.getArea());

        // set rectTwo's position
        rectTwo.setOrigin(originOne);

        // display rectTwo's position
        System.out.println("X Position of rectTwo: " + rectTwo.getOrigin().getX());
        System.out.println("Y Position of rectTwo: " + rectTwo.getOrigin().getY());

        // move rectTwo and display its new position
        rectTwo.move(40, -20);
        System.out.println("X Position of rectTwo: " + rectTwo.getOrigin().getX());
        System.out.println("Y Position of rectTwo: " + rectTwo.getOrigin().getY());
    }
}
```

Στο πρόγραμμα ορίζονται στη μέθοδο main τα εξής

```
Point originOne = new Point(23, 94);

Rectangle rectOne = new Rectangle(100, 200,
 originOne);

Rectangle rectTwo = new Rectangle(50, 100);
```

Το παραπάνω μπορεί να γραφεί και ως εξής:

```
Point originOne;
Rectangle rectOne, rectTwo;

originOne = new Point(23, 94);
rectOne = new Rectangle(100, 200, originOne);
rectTwo = new Rectangle(50, 100);
```

Οι πρώτες δύο γραμμές ορίζουν τις μεταλητές **originOne**, **rectOne**, **rectTwo**. Οι μεταβλητές δυνητικά δείχνουν σε τύπους δεδομένων **Point**, **Rectangle**.

Προς το παρόν όμως το περιεχόμενο των μεταβλητών είναι απροσδιόριστο (*ο compiler αρχικοποεί τις μεταβλητές αυτές στην τιμή null*).

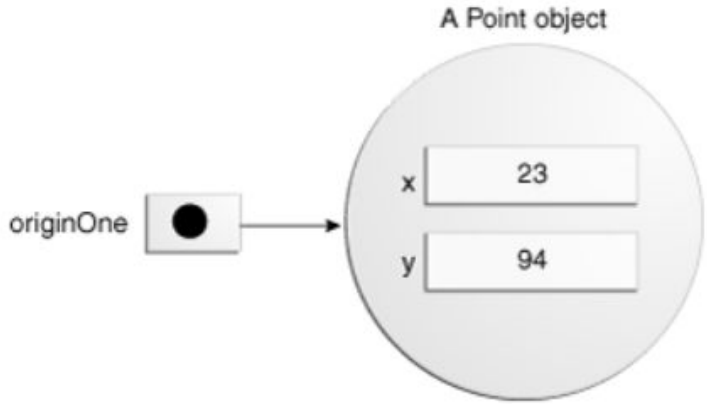
Σε αναλογία με την γλώσσα C, φανταστείτε τις μεταβλητές αυτές ως pointers που δεν είναι αρχικοποιημένοι σε κάποια υφιστάμενη διεύθυνση μνήμης. Η τιμή τους είναι απροσδιόριστη και η προσπάθεια να γράψουμε στη διεύθυνση μνήμης όπου δείχνουν θα προκαλέσει Segmentation Fault.

Η παρακάτω εικόνα δείχνει την ύπαρξη ενός μη αρχικοποιημένου δείκτη.

Προκειμένου να αρχικοποιηθούν οι μεταβλητές απαιτούνται δύο βήματα:

1. Η δέσμευση της απαραίτητης μνήμης, ώστε να μπορούμε να αποθηκεύσουμε τα δεδομένα που προδιαγράφονται από τον συγκεκριμένο τύπο (κλάση).
2. Η αρχικοποίηση των επιμέρους μεταβλητών των αντικειμένων που ορίζονται απο την εκάστοτε κλάση.

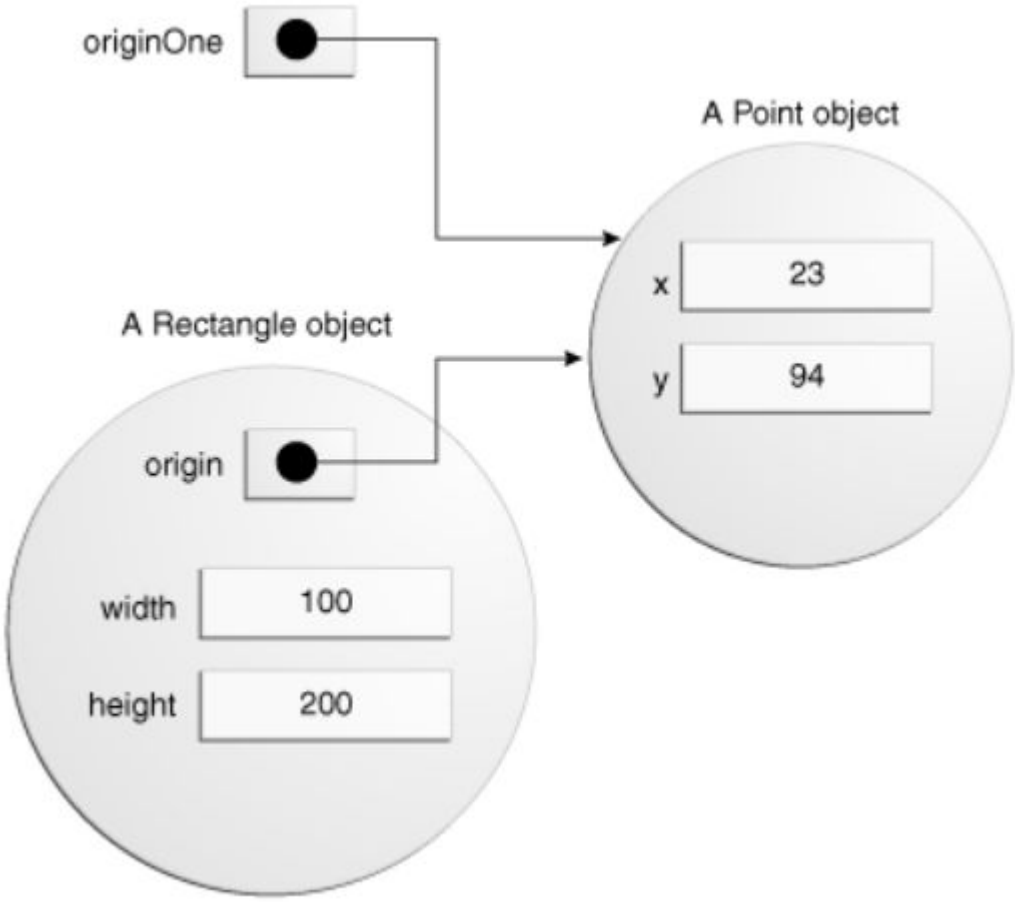
Η παρακάτω εικόνα δείχνει τον δείκτη μετά την δέσμευση της απαιτούμενης μνήμης και την αρχικοποίηση των δεδομένων (μεταβλητών) του αντικειμένου originOne.



Ο κώδικας

```
rectOne = new Rectangle(100, 200, originOne);
```

δηλώνει την δέσμευση μνήμης για ένα αντικείμενο τύπου Rectangle όπου το σημείο της αρχής του είναι το σημείο originOne.



Ο τελεστής new

Προκειμένου να δημιουργηθούν νέα αντικείμενα χρησιμοποιείται ο τελεστής **new**. Ο τελεστής **new** χρησιμοποιείται με τον κατασκευαστή της κλάσης προκειμένου να κάνει τα εξής:

1. Δέσμευση της απαραίτητης μνήμης για τη δημιουργία του αντικειμένου. Η αρχικά ορισμένη μεταβλητή δείχνει πλέον στην περιοχή μνήμης που έχει δεσμευτεί.
2. Αρχικοποίηση των εσωτερικών μεταβλητών (πεδίων) του αντικειμένου με βάση τον κώδικα του κατασκευαστή και τις αρχικές παραμέτρους.
   a. Εάν δεν έχει οριστεί κατασκευαστής τότε ο τελεστής **new** καλείται με χρήση του **default** κατασκευαστή (***default constructor***) που δεν έχει ορίσματα (π.χ. `MyClass obj = new MyClass()`, όπου για την κλάση **MyClass** δεν έχει οριστεί κανένας κατασκευαστής).

Κατά την χρήση primitive τύπων δεδομένων σε ένα πρόγραμμα (int, float, double) δεν απαιτείται η χρήση του τελεστή **new**. Για τους συγκεκριμένους τύπους δεδομένων είναι γνωστό εκ των προτέρων το εύρος μνήμης που απαιτούν.

Αντίθετα, για τα αντικείμενα των κλάσεων ο *compiler* δεν μπορεί να γνωρίζει εκ των προτέρων το μέγεθος τους, καθώς με την σειρά τους αυτά μπορεί να περιέχουν άλλα αντικείμενα. Για τον λόγο αυτό, για τα αντικείμενα δεσμεύεται πάντοτε η απαραίτητη μνήμη δυναμικά.

Χρήση πεδίων και μεθόδων

Χρήση πεδίων των αντικειμένων

Τα πεδία ενός αντικειμένου είναι προσβάσιμα με χρήση του ονόματος του αντικειμένου, μία τελεία '.' και το όνομα του πεδίου.

```
Rectangle rect = new Rectangle(10,20);

System.out.println("Rectangle dimensions
are " + rect.width + ", " + rect.height);
```

Χρήση μεθόδων των αντικειμένων

Σε αναλογία με τα πεδία οι μέθοδοι ενός αντικειμένου είναι προσβάσιμες μέσω του ονόματος του αντικειμένου, μία τελεία '.' και το όνομα της μεθόδου.

```
Rectangle rect = new Rectangle(10,20);

System.out.println("Rectangle dimensions
are " + rect.getWidth() + ", " +
rect.getHeight() );
```

Περιορισμός της πρόσβασης με τους προσδιοριστές public και private

- Οι προσδιοριστές **public** και **private** αφορούν την προσβασιμότητα των πεδίων ή των μεθόδων της κλάσης από άλλες κλάσεις.
- Όταν ένα πεδίο ή μία μέθοδος ορίζεται ως **public** τότε είναι προσβάσιμο/η από οποιαδήποτε άλλη κλάση. Όταν ένα πεδίο ή μια μέθοδος ορίζεται ως **private** τότε είναι προσβάσιμο/η μόνο εντός της κλάσης στην οποία έχει οριστεί.
- Με βάση τα παραπάνω η κλάση Rectangle δεν έχει απευθείας πρόσβαση στα πεδία x και y του αντικειμένου origin που ανήκει στην κλάση Point, καθώς τα x και y έχουν οριστεί ως private.

```
private int x;
private int y;
```

- Σε αυτή την περίπτωση, ο μόνος τρόπος να προσπελαστούν τα δεδομένα της κλάσης Point από την κλάση Rectangle είναι μέσω των μεθόδων

```
public int getX()
public int setX(int xPos)
public int getY()
public int setY(int yPos)
```

Περιορισμός της πρόσβασης - Παράδειγμα

Στην κλάση **CreateObjectDemo**, αλλάζουμε τον τρόπο πρόσβασης στα πεδία x,y της κλάσης **Point**, ώστε η πρόσβαση να γίνεται απευθείας με χρήση των πεδίων και όχι των βοηθητικών μεθόδων.

- Προσπαθήστε να μεταγλωττίσετε τον παρακάτω κώδικα και παρατηρήστε τα λάθη που εμφανίζει ο μεταγλωττιστής.
- Στη συνέχεια αλλάξτε όλα τα πεδία των εμπλεκόμενων κλάσεων από *private* σε *public*. Τι παρατηρείτε;

```
public class CreateObjectDemo {

    public static void main(String[] args) {

        // Declare and create a point object and two rectangle objects.
        Point originOne = new Point(23, 94);
        Rectangle rectOne = new Rectangle(100, 200, originOne);
        Rectangle rectTwo = new Rectangle(50, 100);

        // display rectOne's width, height, and area
        System.out.println("Width of rectOne: " + rectOne.getWidth() );
        System.out.println("Height of rectOne: " + rectOne.getHeight() );
        System.out.println("Area of rectOne: " + rectOne.getArea());

        // set rectTwo's position
        rectTwo.setOrigin(originOne);

        // display rectTwo's position
        System.out.println("X Position of rectTwo: " + rectTwo.origin.x);
        System.out.println("Y Position of rectTwo: " + rectTwo.origin.y);

        // move rectTwo and display its new position
        rectTwo.move(40, -20);
        System.out.println("X Position of rectTwo: " + rectTwo.origin.x);
        System.out.println("Y Position of rectTwo: " + rectTwo.origin.y);
    }
}
```

Απόκρυψη της εσωτερικής υλοποίησης

Ένα από τα βασικά χαρακτηριστικά του Αντικειμενοστραφούς Προγραμματισμούς είναι η απόκρυψη των δεδομένων και της εσωτερικής υλοποίησης των κλάσεων στις υπόλοιπες κλάσεις που τις χρησιμοποιούν.

Κατά κανόνα, συνηθίζουμε να παρέχουμε ένα σύνολο από μεθόδους, τις οποίες μπορούν να καλούν οι υπόλοιπες κλάσεις προκειμένου να χρησιμοποιήσουν την κλάση.

Ο λόγος που συνήθως αποκρύπτουμε την εσωτερική υλοποίηση είναι ότι αν δεν γίνεται απευθείας χρήση των εσωτερικών δεδομένων μίας κλάσης ή των εσωτερικών μεθόδων αυτής, τότε η εσωτερική υλοποίηση της κλάσης μπορεί να αλλάξει, χωρίς η αλλαγή να επηρεάσει τις υπόλοιπες κλάσεις που χρησιμοποιούν την συγκεκριμένη κλάση.

Η παραπάνω πρακτική συναντάται στην βιβλιογραφία ως information hiding.

Για κάθε πεδίο των κλάσεων **Point** και **Rectangle** έχουμε ορίσει συναρτήσεις λήψης της τιμής των πεδίων της κλάσης (γνωστές ως getters ή accessors), αλλά και συναρτήσεις για τον ορισμό ή την μεταβολή της τιμής τους (γνωστές ως setters ή mutators).

Garbage Collector

Είδαμε ότι η JAVA επιτρέπει στον προγραμματιστεί να ορίσει τα αντικείμενα που επιθυμεί και δεσμεύει την μνήμη για αυτά μέσω του τελεστή **new**.

Περιοδικά το **JVM** κοιτάει εάν υπάρχει δεσμευμένη μνήμη για αντικείμενα στα οποία δεν υφίστανται πλέον μεταβλητές που δείχνουν σε αυτά, δηλαδή δεν υφίστανται references προς αυτά. Σε αυτές τις περιπτώσεις, ελευθερώνεται η μνήμη που έχει δεσμευτεί για τα αντικείμενα αυτή της κατηγορίας.

Παράλληλα η αποδέσμευση της μνήμης και η διαγραφή των αντικειμένων πιθανόν συνεπάγεται ότι και άλλα αντικείμενα δεν διαθέτουν πια αναφορές προς αυτά κ.ο.κ. Η διαδικασία συνεχίζεται μέχρι να αποδεσμευτεί όλη η δυναμικά δεσμευμένη μνήμη.

Η ύπαρξη του Garbage Collection προϋποθέτει ότι ένα παράλληλο νήμα, τρέχει περιοδικά και ελευθερώνει τη μνήμη που δεν χρησιμοποιείται πλέον. Το παραπάνω πιθανόν να εισάγει καθυστερήσεις σε εφαρμογές με συγκεκριμένους περιορισμούς ως προς τον χρόνο εκτέλεσης τους.