Φροντιστήριο JAVA

Οντοκεντρικός Προγραμματισμός (lect 1)

Ελένη Βογιατζάκη
evoyiatzaki@ceid.upatras.gr

Διαδικαστικά Μαθήματος

Θεωρία – Φροντιστήριο - Εργαστήριο

C

Java

/** My first program */

```
#include <stdio.h>
int main()

{
printf("Hello World\n");
printf(" Hello class 2020  \n ");
return 0;


}
```

```
public class HelloWorld
{    public static void main(String []args)
{
System.out.println("Hello World  \n");
 System.out.println("Hello class 2020  \n");
 }
}
```

Παραδείγματα από Primitive data types:

```
int myNum;              // Integer
char myLetter = 'D';    // Character
```

….

….

Δεν μας αρκούν ….

Δημιουργούμε δικές μας «κατασκευές»

Τι κάνουμε με τα primitive data types:

```
int a,b,answer;              //Δήλωση μεταβλητών
a = 5;                       // Εισαγωγή τιμών
b=3;
answer = addition (a,b); // Συναρτήσεις και μεταβλητές
……….
Function addition   //κώδικας function
…….
…….
```

Όμως μας φτάνουν τα primitive data types?

“κατασκευή” StudentData σε δύο “εκδόσεις”

```c
#include <stdio.h>
/* Created a structure here. The name of the structure is StudentData. */

struct StudentData
{
int stu_id;
int stu_grade; };

int main()
{
/* student is the variable of structure StudentData*/
struct StudentData student;

/*Assigning the values of each struct member here*/
student.stu_id = 1234;
student.stu_grade = 9;

/* Displaying the values of struct members */
printf("\nStudent Id is: %d", student.stu_id);
printf("\nStudent grade is: %d", student.stu_grade);

return 0; }
```

Δημιουργία μίας «κατασκευής»

Εισαγωγή τιμών

Συναρτήσεις

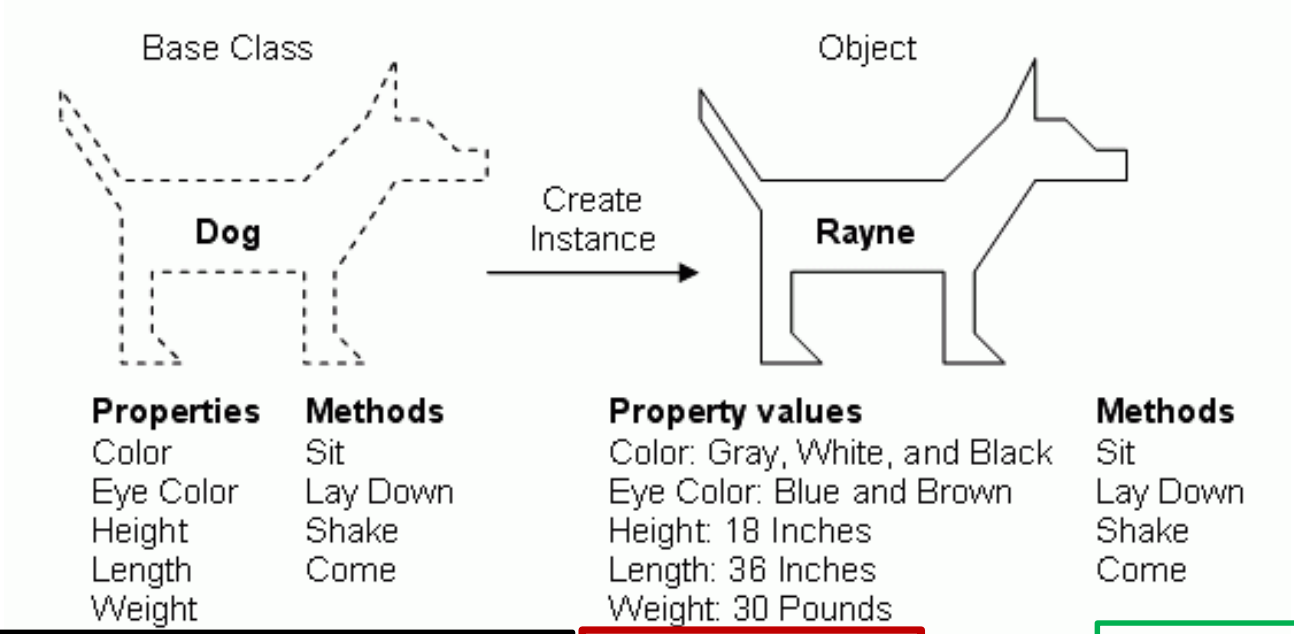
```c
public class Studentdata
{
  /* Created a structure here. The name of the structure is StudentData.
*/
 int stu_id;
 int stu_grade;
   /*Constructor for objects of class studentdata
    */
   public Studentdata()
   {
  /*Assigning the values of each struct member here*/
      stu_id = 1234;
      stu_grade = 9;
   }

   public  void Printdata ()
   {
    /* Displaying the values of struct members */
   System.out.println("Student Id is: "+ stu_id);
   System.out.println("Student grade is: "+ stu_grade);
   }

   public static void main ()
   {
   Studentdata st = new Studentdata();
   st.Printdata();

   }
}
```

Αντικειμενοστρεφής τρόπος σκέψης



Δημιουργία μίας «κατασκευής»

Συναρτήσεις

Δημιουργία αντικειμένου

Εισαγωγή τιμών

Συναρτήσεις

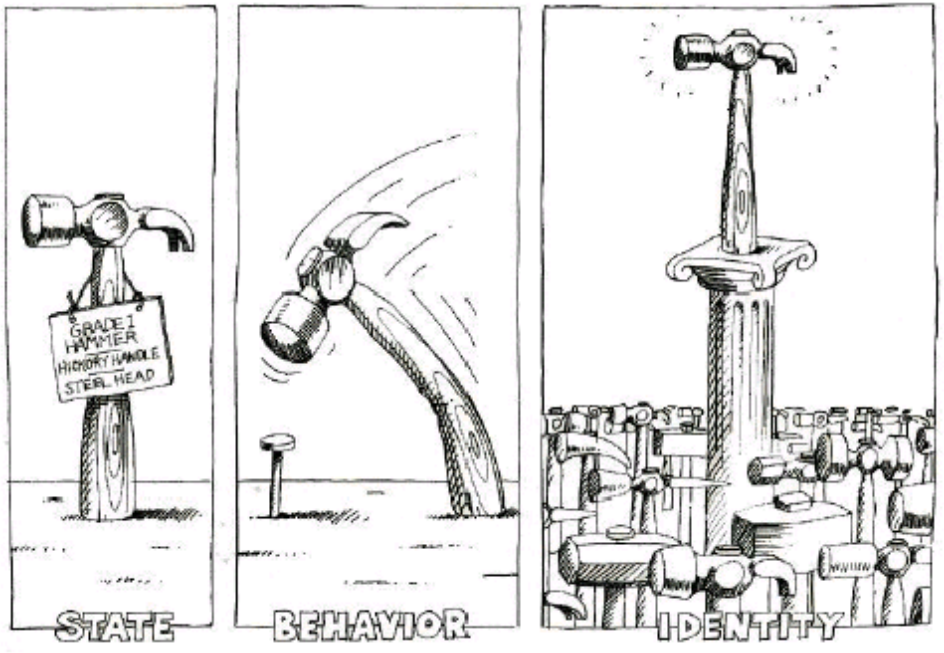
Βασικές αρχές αντικειμενοστρέφειας (αρχές του Allan Kay)

- Τα πάντα είναι αντικείμενα
- Κάθε αντικείμενο έχει δικιά του μνήμη
- Κάθε αντικείμενο έχει ένα συγκεκριμένο τύπο
  - Τύπος = Κλάση
- Τα αντικείμενα επικοινωνούν μέσω μηνυμάτων
- Αντικείμενα του ίδιου τύπου μπορούν να δεχτούν τα ίδια μηνύματα
  - Δηλαδή μπορούν να εκτελούν τις ίδιες λειτουργίες
- Ένα πρόγραμμα είναι μια συλλογή από αντικείμενα όπου το ένα λέει στο άλλο τι να κάνει

Αντικείμενο

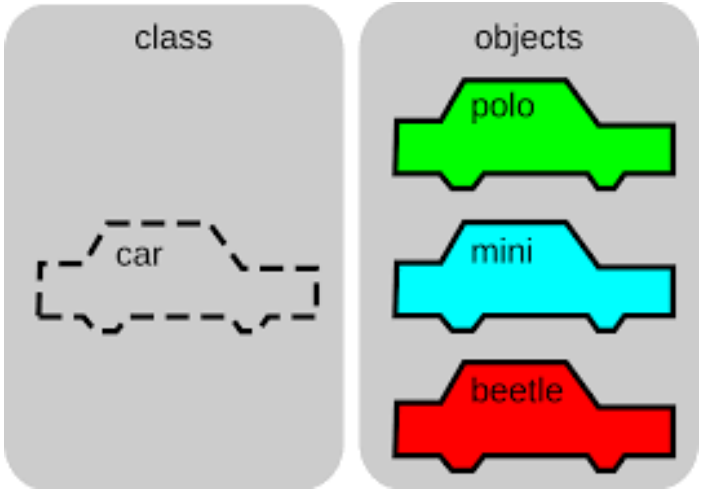
- Ένα αντικείμενο στον κώδικα αναπαριστά μια οντότητα (έννοια) και έχει:

- Μια κατάσταση, η οποία ορίζεται από ορισμένα χαρακτηριστικά
- Μια συμπεριφορά, η οποία ορίζεται από ορισμένες ενέργειες που μπορεί να εκτελέσει το αντικείμενο
- Μια ταυτότητα που το ξεχωρίζει από τα υπόλοιπα.



Παραδείγματα: ένας άνθρωπος, ένα πράγμα, ένα μέρος, μια υπηρεσία

Κλάσεις



- Κλάση: Μια αφηρημένη περιγραφή αντικειμένων με κοινά χαρακτηριστικά και κοινή συμπεριφορά
  - Ένα καλούπι που παράγει αντικείμενα
  - Ένα αντικείμενο είναι ένα στιγμιότυπο μίας κλάσης.

- Π.χ., η κλάση φοιτητής έχει τα γενικά χαρακτηριστικά (όνομα, βαθμοί) και τη συμπεριφορά print
  - Ο φοιτητής Χ είναι ένα αντικείμενο της κλάσης φοιτητής

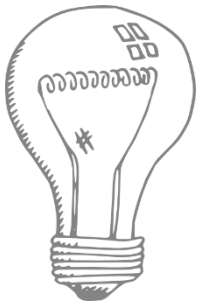
- Η κλάση Car έχει τα χαρακτηριστικά (brand, color) και τη συμπεριφορά (drive, stop)
  - Το αυτοκίνητο ΑΧΧ2013 είναι ένα αντικείμενο της κλάσης Car με κατάσταση τα χαρακτηριστικά (BMW, red)

Πρακτικά στον κώδικα

- Μία κλάση Κ ορίζεται από
  - Κάποιες μεταβλητές τις οποίες ονομάζουμε πεδία
  - Κάποιες συναρτήσεις που τις ονομάζουμε μεθόδους.
    - Οι μέθοδοι «βλέπουν» τα πεδία της κλάσης

- Ένα αντικείμενο ορίζεται ως μια μεταβλητή τύπου Κ
  - Στην Java (όπως και στις περισσότερες γλώσσες) όλες οι μεταβλητές έχουν ένα τύπο (“strongly typed”).
  - Το αντικείμενο που δημιουργείται παίρνει κάποιες τιμές στα πεδία της κλάσης και καταλαμβάνει κάποιο χώρο στη μνήμη.
    - Έτσι μετατρέπεται σε ένα φυσικό αντικείμενο με μοναδική ταυτότητα.

Διαχειρίζομαι τους λαμπτήρες σε ένα σπίτι



Θέλουμε να κάνουμε ένα πρόγραμμα που να διαχειρίζεται τους λαμπτήρες (φώτα) σε διάφορα δωμάτια και θα υλοποιεί και ένα dimmer

Αντικείμενα:

bedroomLight
kitchenLight

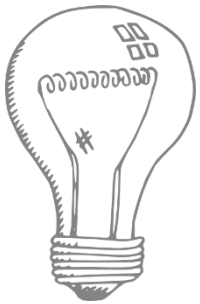
Light	Όνομα κλάσης
intensity	Πεδία κλάσης
on() off() dim() brighten()	Μέθοδοι κλάσης

Τα αντικείμενα δημιουργούνται σε άλλο σημείο του κώδικα το οποίο καλεί και τις μεθόδους

Light bedroomLight
Light kitchenLight

Η κλήση μιας μεθόδου σε ένα αντικείμενο μερικές φορές λέγεται και πέρασμα μηνύματος
bedroomLight.on()

Διαχειρίζομαι τους λαμπτήρες σε ένα σπίτι



Θέλουμε να κάνουμε ένα πρόγραμμα που να διαχειρίζεται τους λαμπτήρες (φώτα) σε διάφορα δωμάτια και θα υλοποιεί και ένα dimmer

Αντικείμενα:

bedroomLight
kitchenLight

Τα αντικείμενα δημιουργούνται σε άλλο σημείο του κώδικα το οποίο καλεί και τις μεθόδους

Light bedroomLight
Light kitchenLight

Light	Όνομα κλάσης
intensity	Πεδία κλάσης
on() off() dim() brighten()	Μέθοδοι κλάσης

Η κλήση μιας μεθόδου σε ένα αντικείμενο μερικές φορές λέγεται και πέρασμα μηνύματος
bedroomLight.on()

Πλεονεκτήματα Object Orientation

- Τα αντικείμενα και οι κλάσεις μοντελοποιούν με φυσικό τρόπο τα αντικείμενα του κόσμου
- Τα πλεονεκτήματα είναι ότι αυτό κάνει τον κώδικα
  - πιο ευανάγνωστο
  - πιο τμηματοποιημένο
  - και πιο εύκολο να συντηρηθεί.

Παράδειγμα

- Θέλουμε να προσομοιώσουμε ένα αυτοκίνητο το οποίο κινείται πάνω σε μία ευθεία. Αρχικά ξεκινάει από τη θέση μηδέν. Σε κάθε χρονική στιγμή κινείται τυχαία κατά μία θέση είτε αριστερά είτε δεξιά. Σε κάθε βήμα τυπώνεται μια κουκίδα που δείχνει τη θέση του.

- Πώς θα λύσουμε αυτό το πρόβλημα?
  - Τι κλάσεις και τι αντικείμενα θα ορίσουμε?
  - Τι πεδία και τι μεθόδους θα έχουν?

Αντικειμενοστρεφής Σχεδίαση

- Οι οντότητες/έννοιες στον ορισμό του προβλήματος γίνονται κλάσεις και ορίζονται τα αντικείμενα που αναφέρονται στο πρόβλημα.
- Τα ρήματα γίνονται μέθοδοι
- Τα χαρακτηριστικά των αντικειμένων γίνονται πεδία
  - Τα πεδία μπορεί να είναι κι αυτά αντικείμενα.

- Δεν υπάρχει ένας μοναδικός τρόπος να μοντελοποιήσετε ένα πρόβλημα. Συνήθως όμως υπάρχει ένας που είναι καλύτερος από τους άλλους.

Παράδειγμα

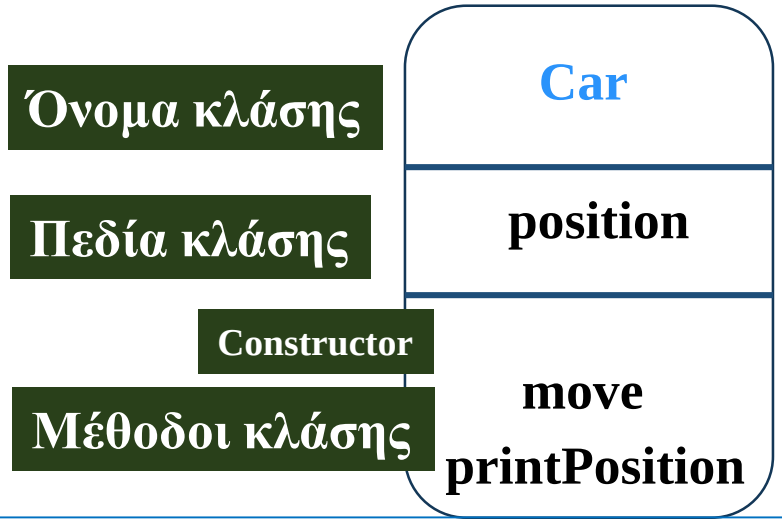
- Θέλουμε να προσομοιώσουμε ένα αυτοκίνητο το οποίο κινείται πάνω σε μία ευθεία. Αρχικά ξεκινάει από τη θέση μηδέν. Σε κάθε χρονική στιγμή κινείται τυχαία κατά μία θέση είτε αριστερά είτε δεξιά. Σε κάθε βήμα τυπώνεται μια κουκίδα που δείχνει τη θέση του.

- Πώς θα λύσουμε αυτό το πρόβλημα?
  - Τι κλάσεις και τι αντικείμενα θα ορίσουμε?
  - Τι πεδία και τι μεθόδους θα έχουν?

Στην πραγματικότητα ενώ μιλάω για αυτοκίνητο με ενδιαφέρει η θέση του.

Υλοποίηση

Αν έχω δύο αυτοκίνητα?



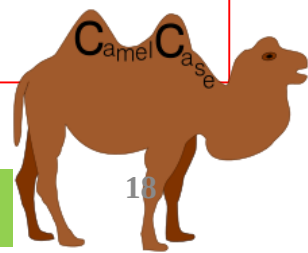
Πρόγραμμα
Δημιούργησε το αντικείμενο myCar τύπου Car

```
Car myCar = new Car()

myCar.printPosition()
For i = 1…10
    myCar.move()
    myCar.printPosition()
```

Programming Style

- Τα ονόματα των κλάσεων ξεκινάνε με κεφαλαίο, τα ονόματα των πεδίων, μεθόδων και αντικειμένων με μικρό.
- Χρησιμοποιούμε ολόκληρες λέξεις (και συνδυασμούς τους) για τα ονόματα
  - Δεν πειράζει αν βγαίνουν μεγάλα ονόματα
- Χρησιμοποιούμε το **CamelCase** Style
  - Όταν για ένα όνομα έχουμε πάνω από μία λέξη, τις συνενώνουμε και στο σημείο συνένωσης κάνουμε το πρώτο γράμμα της λέξης κεφαλαίο
  - printPosition όχι print_position
- Χρησιμοποιούμε **κεφαλαία** και '_' για τις **σταθερές**.



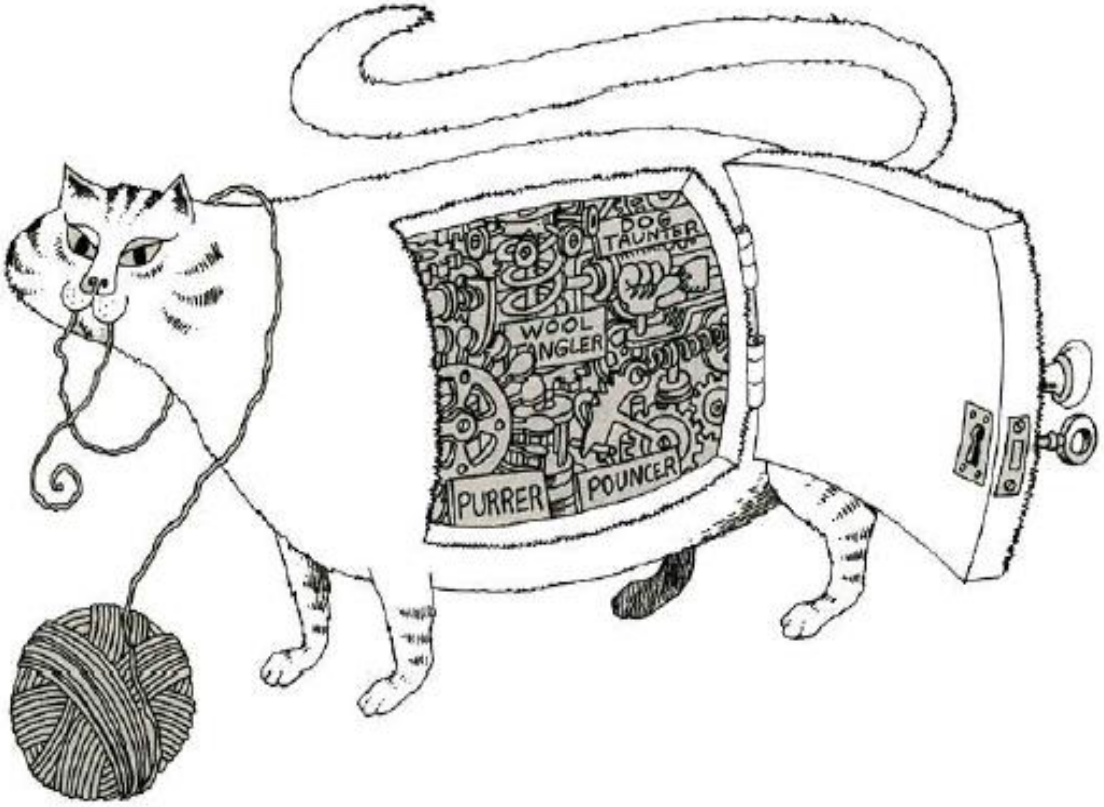
Λείπει κάτι?

Αρχικοποίηση?

Απόκρυψη - Ενθυλάκωση

- Στο πρόγραμμα που κάναμε πριν δεν είχαμε πρόσβαση στην θέση του αυτοκινήτου
  - Μόνο οι μέθοδοι της κλάσης μπορούν να την αλλάξουν.
  - Γιατί?
    - Αν μπορούσε να αλλάζει σε πολλά σημεία στον κώδικα τότε κάποιες άλλες μέθοδοι θα μπορούσαν να το αλλάξουν και να δημιουργηθεί μπέρδεμα
    - Τώρα αλλάζει μόνο όταν είναι λογικό να αλλάξει – όταν κινηθεί το όχημα.
- Κάποιος που χρησιμοποιεί τις μεθόδους της κλάσης δεν ξέρει πως υλοποιούνται, γνωρίζει μόνο τι κάνουν και πως μπορεί να τις καλέσει
- Αυτή η αρχή λέγεται Ενθυλάκωση – Απόκρυψη Πληροφορίας

Ενθυλάκωση



- **Η στεγανοποίηση της κατάστασης και της συμπεριφοράς ώστε οι λεπτομέρειες της υλοποίησης να είναι κρυμμένες από το χρήστη του αντικειμένου.**

Παράδειγμα

- Τι γίνεται αν στο αρχικό μας παράδειγμα αν εκτός από αυτοκίνητο είχαμε και μία μηχανή? Η μηχανή κινείται με τον ίδιο τρόπο αλλά όταν τυπώνεται η θέση της αντί για κουκίδα (.) τυπώνεται ένα αστέρι (*).

- Τι κλάσεις πρέπει να ορίσουμε?

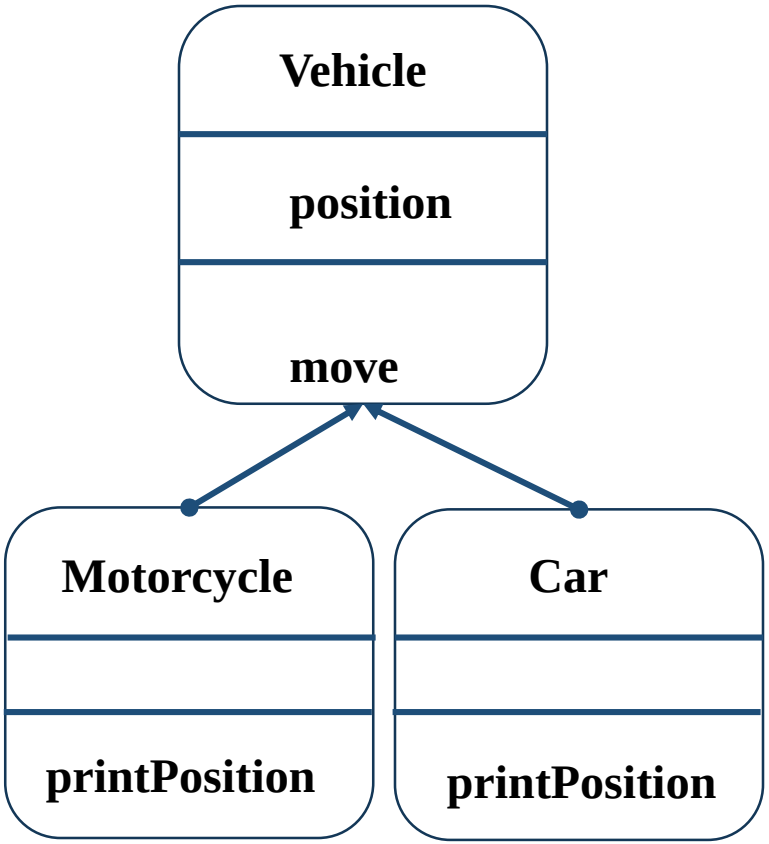
Μία λύση

- Μπορούμε να ορίσουμε ξανά από την αρχή μια καινούρια κλάση για τη μηχανή που να κάνει την ίδια κίνηση με το αυτοκίνητο (ίδια μέθοδο move) αλλά τυπώνεται διαφορετικά (διαφορετική μέθοδο printPosition)

- Τι προβλήματα έχει αυτό?
  - Επανάληψη του κώδικα (μπορεί να είναι μεγάλος και δύσκολος)
  - Πιθανότητα για λάθη
  - Πολύ δύσκολο να γίνουν αλλαγές (θα πρέπει να γίνονται σε δύο σημεία)

Κληρονομικότητα

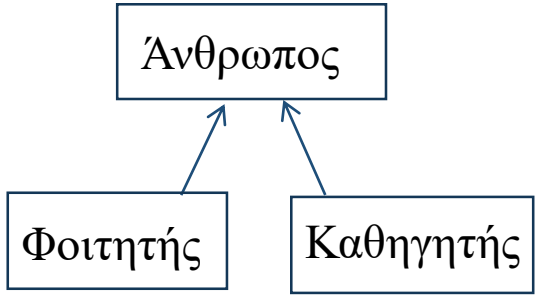
- Ορίζουμε μια κλάση Vehicle (όχημα) η οποία έχει θέση, και έχει κίνηση (μέθοδο move)

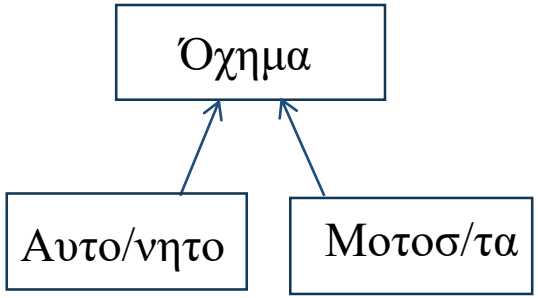



```
Πρόγραμμα
Car myCar = new Car()
Motorcycle myMoto = new Motorcycle()
myCar.printPosition()
myMoto.printPosition()
For i = 1…10
        myCar.move()
        myCar.printPosition()
        myMoto.move()
        myMoto.printPosition()
```

Κληρονομικότητα

- Οι κλάσεις μας επιτρέπουν να ορίσουμε μια ιεραρχία
  - Π.χ., και ο Φοιτητής και ο Καθηγητής ανήκουν στην κλάση Άνθρωπος.
  - Η κλάση Αυτοκίνητο ανήκει στην κλάση Όχημα η οποία περιέχει και την κλάση Μοτοσυκλέτα
- Οι κλάσεις πιο χαμηλά στην ιεραρχία κληρονομούν χαρακτηριστικά και συμπεριφορά από τις ανώτερες κλάσεις
  - Όλοι οι άνθρωποι έχουν όνομα
  - Όλα τα οχήματα έχουν μέθοδο drive, stop.



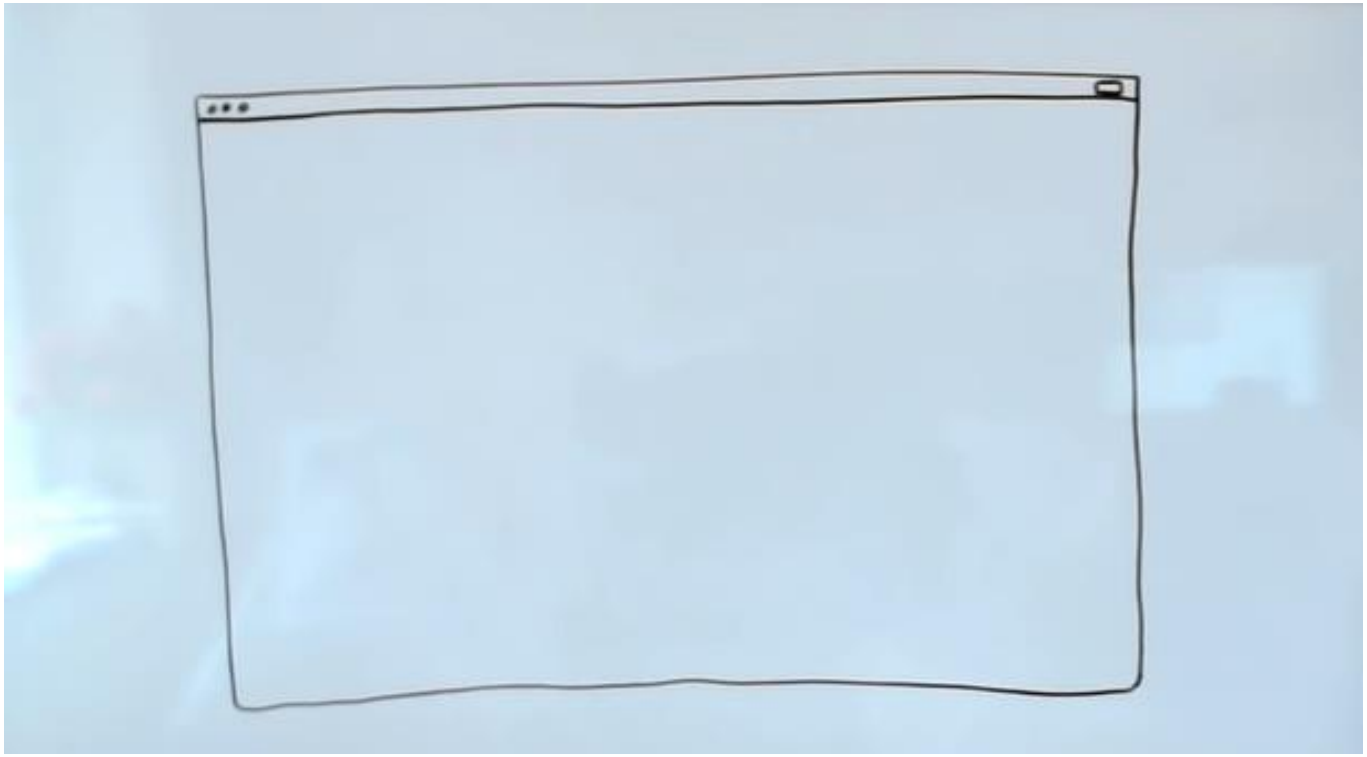


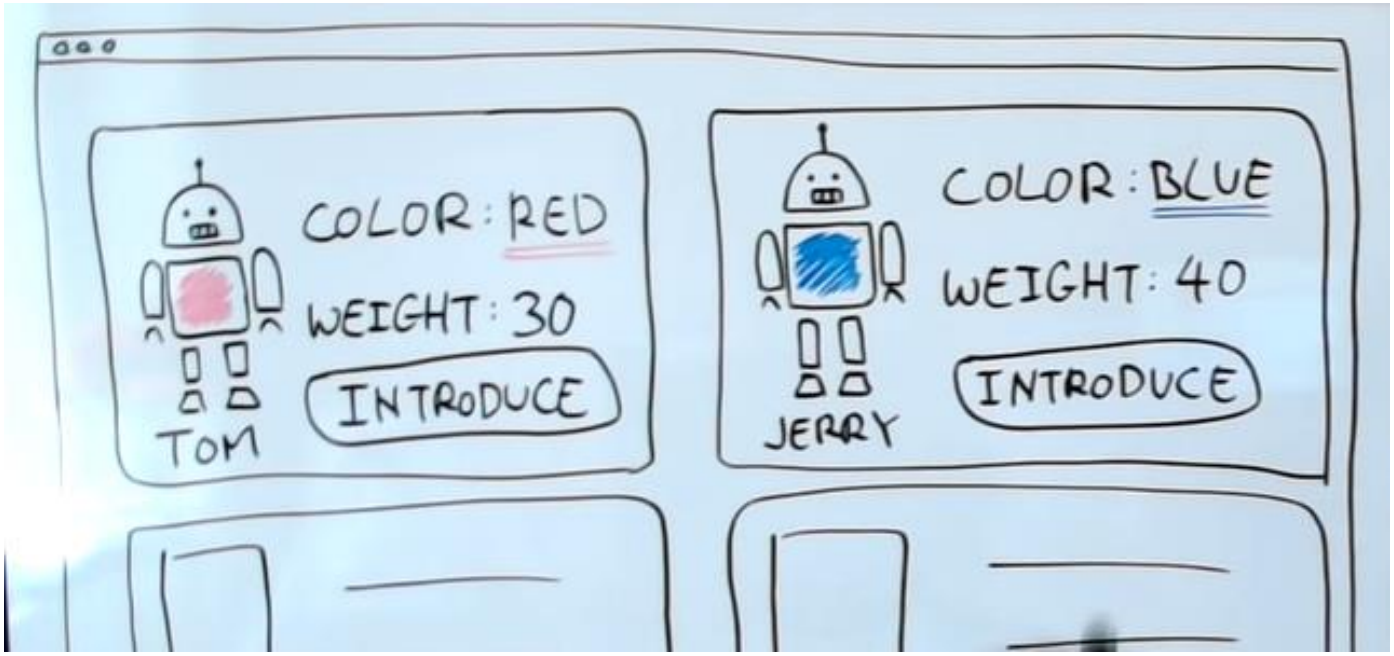
Οι κλάσεις σχηματίζουν ιεραρχία

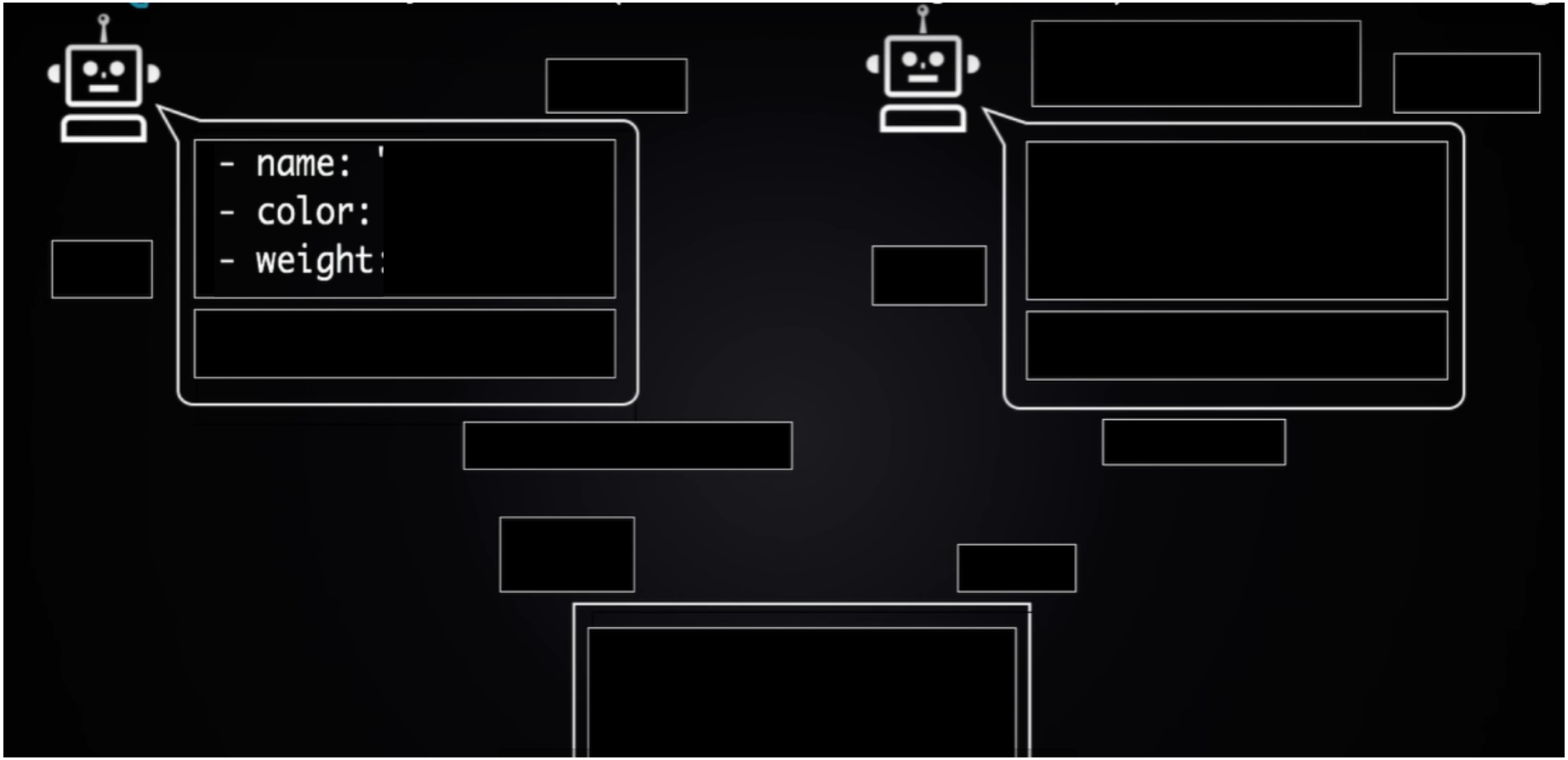
- Οι κλάσεις δομούνται σε μια δενδρική ιεραρχία
- Η κλάση στη ρίζα της ιεραρχίας ονομάζεται Object
- Κάθε κλάση εκτός από την Object, έχει μια υπερκλάση (superclass)
- Μια κλάση μπορεί να έχει πολλούς προγόνους μέχρι την Object
- Όταν ορίζουμε μια κλάση, ορίζουμε την υπερκλάση της
  - Αν δεν ορίσουμε υπερκλάση θεωρείται η Object
- Κάθε κλάση μπορεί να έχει μία ή περισσότερες υποκλάσεις (subclasses)

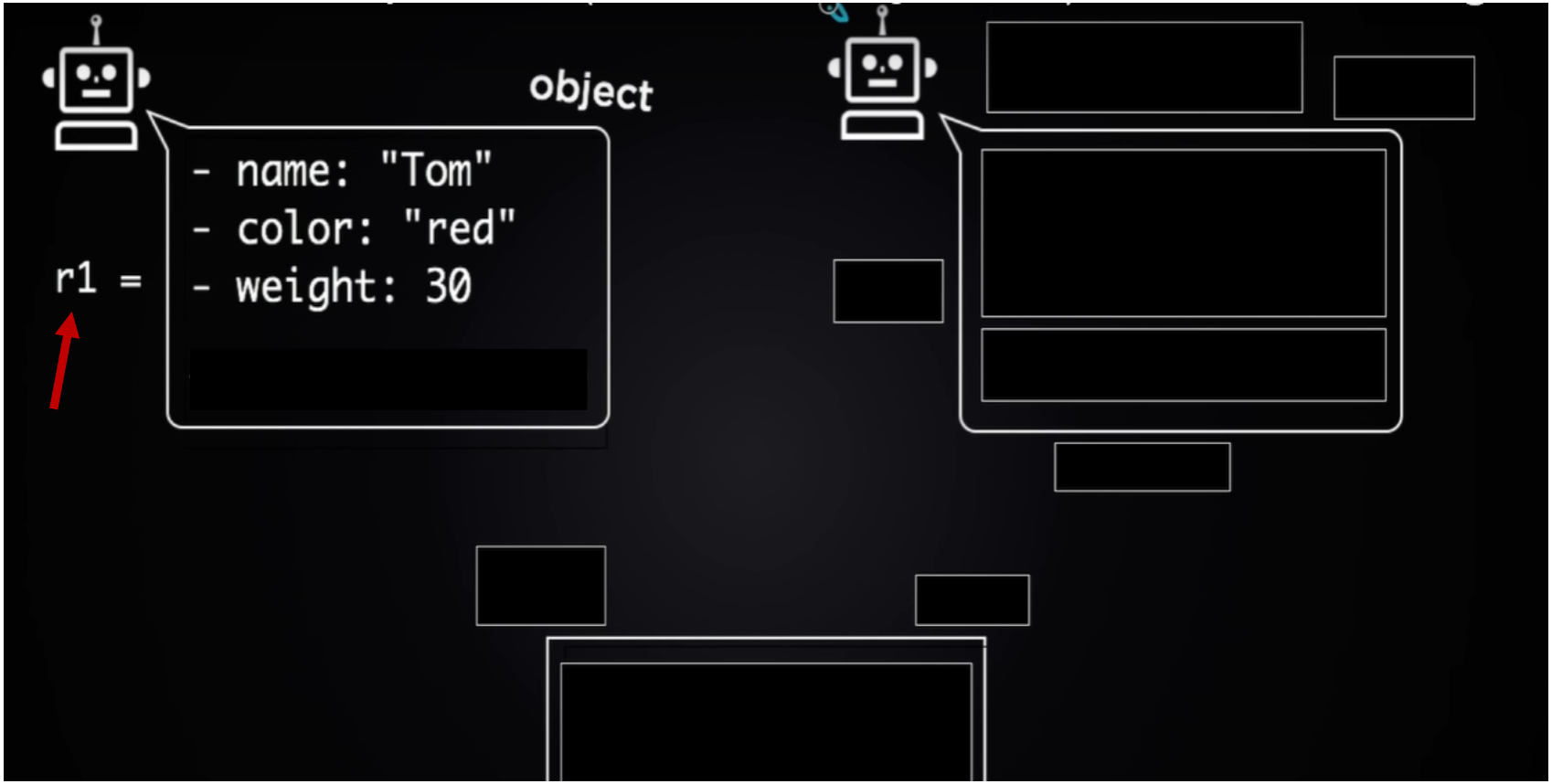
Πολυμορφισμός

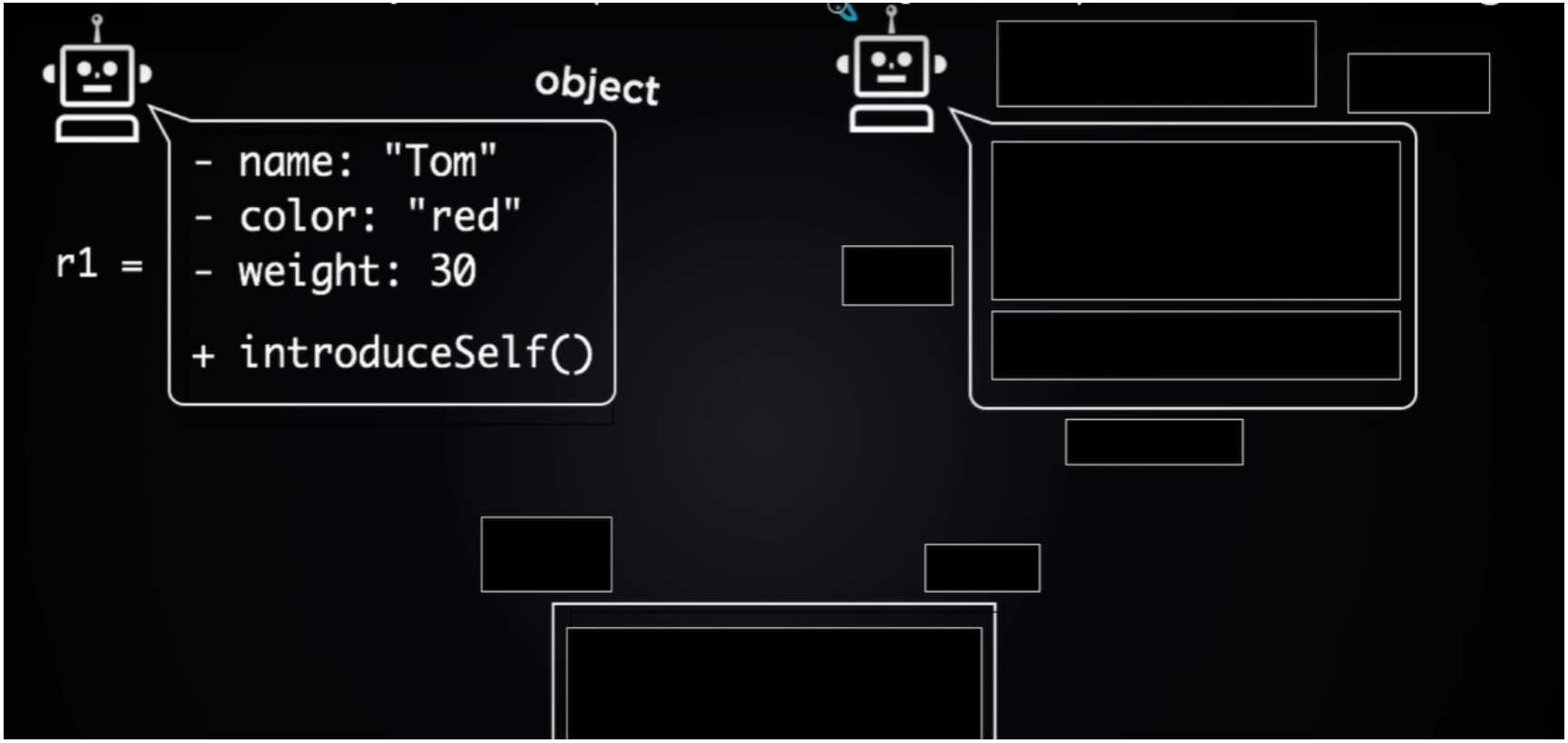
- Κλάσεις με κοινό πρόγονο έχουν κοινά χαρακτηριστικά, αλλά έχουν και διαφορές
  - Π.χ., είναι διαφορετικό το παρκάρισμα για ένα αυτοκίνητο και μια μηχανή
- Ο πολυμορφισμός μας επιτρέπει να δώσουμε μια κοινή συμπεριφορά σε κάθε κλάση (μια μέθοδο park), η οποία όμως υλοποιείται διαφορετικά για αντικείμενα διαφορετικών κλάσεων.
- Μπορούμε επίσης να ορίσουμε αφηρημένες κλάσεις, όπου προϋποθέτουμε μια συμπεριφορά και αυτή πρέπει να υλοποιηθεί σε χαμηλότερες κλάσεις διαφορετικά ανάλογα με τις ανάγκες μας

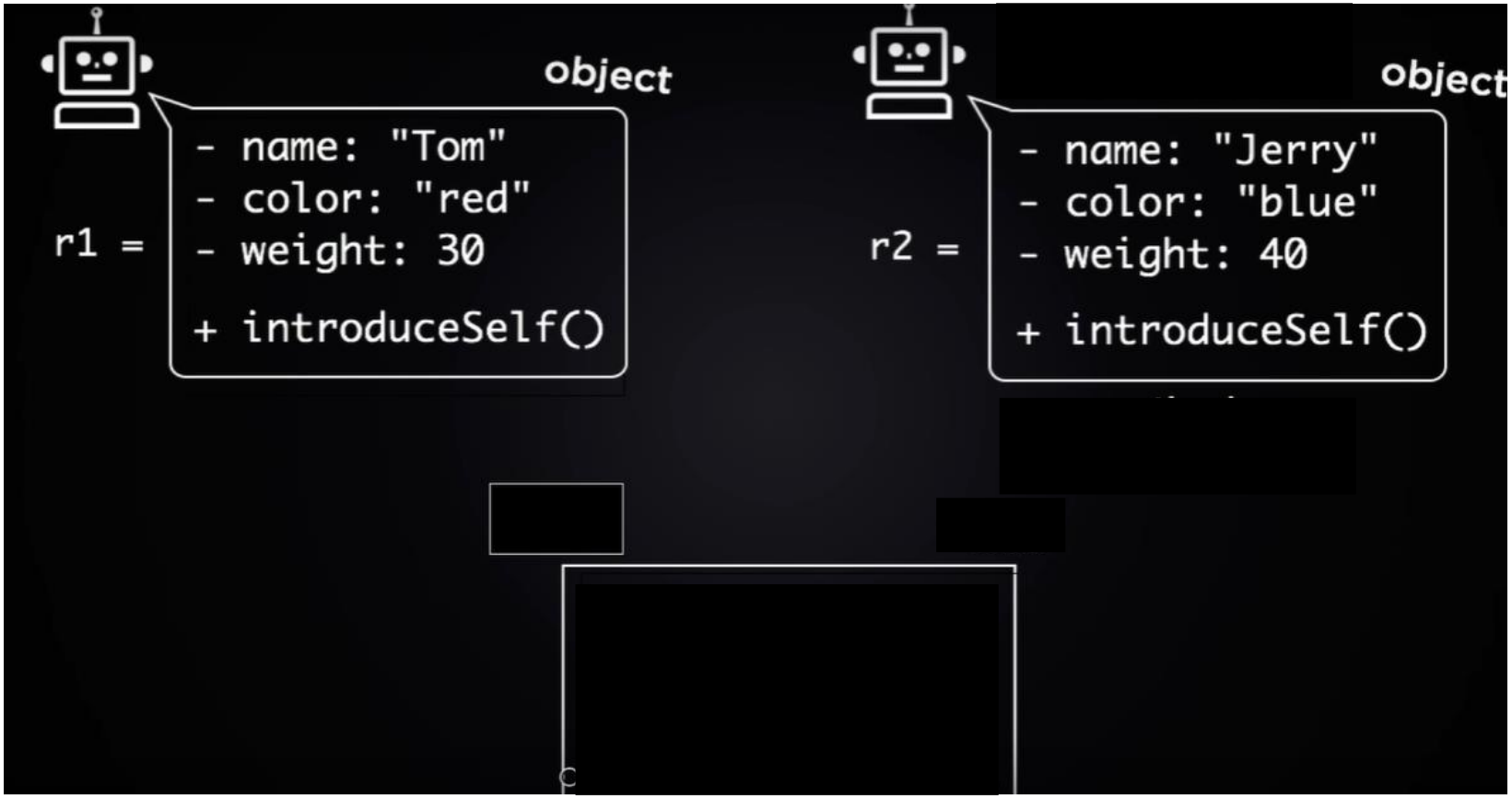


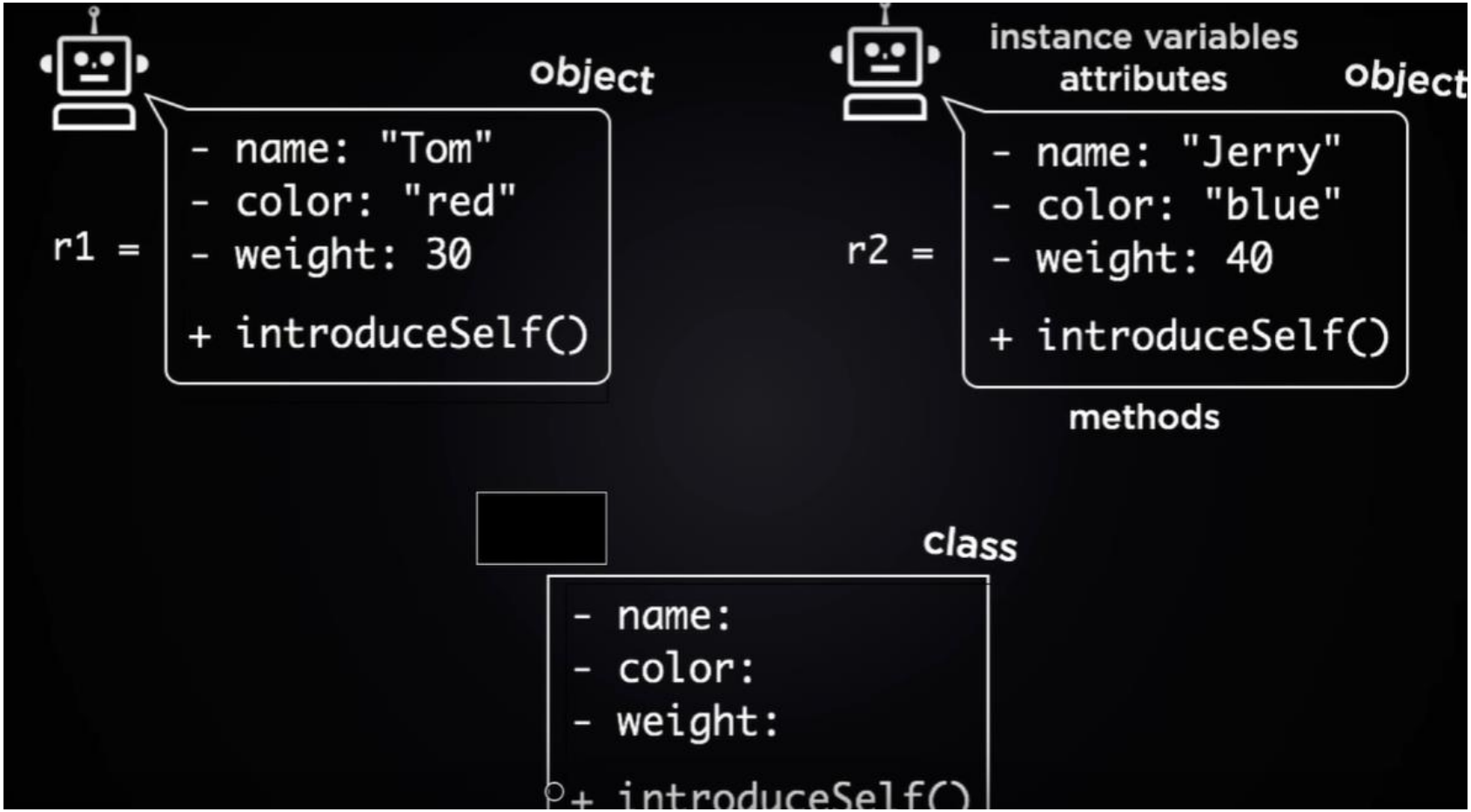

COLOR: RED
WEIGHT: 30
INTRODUCE
TOM
COLOR: BLUE
WEIGHT: 40
INTRODUCE
JERRY

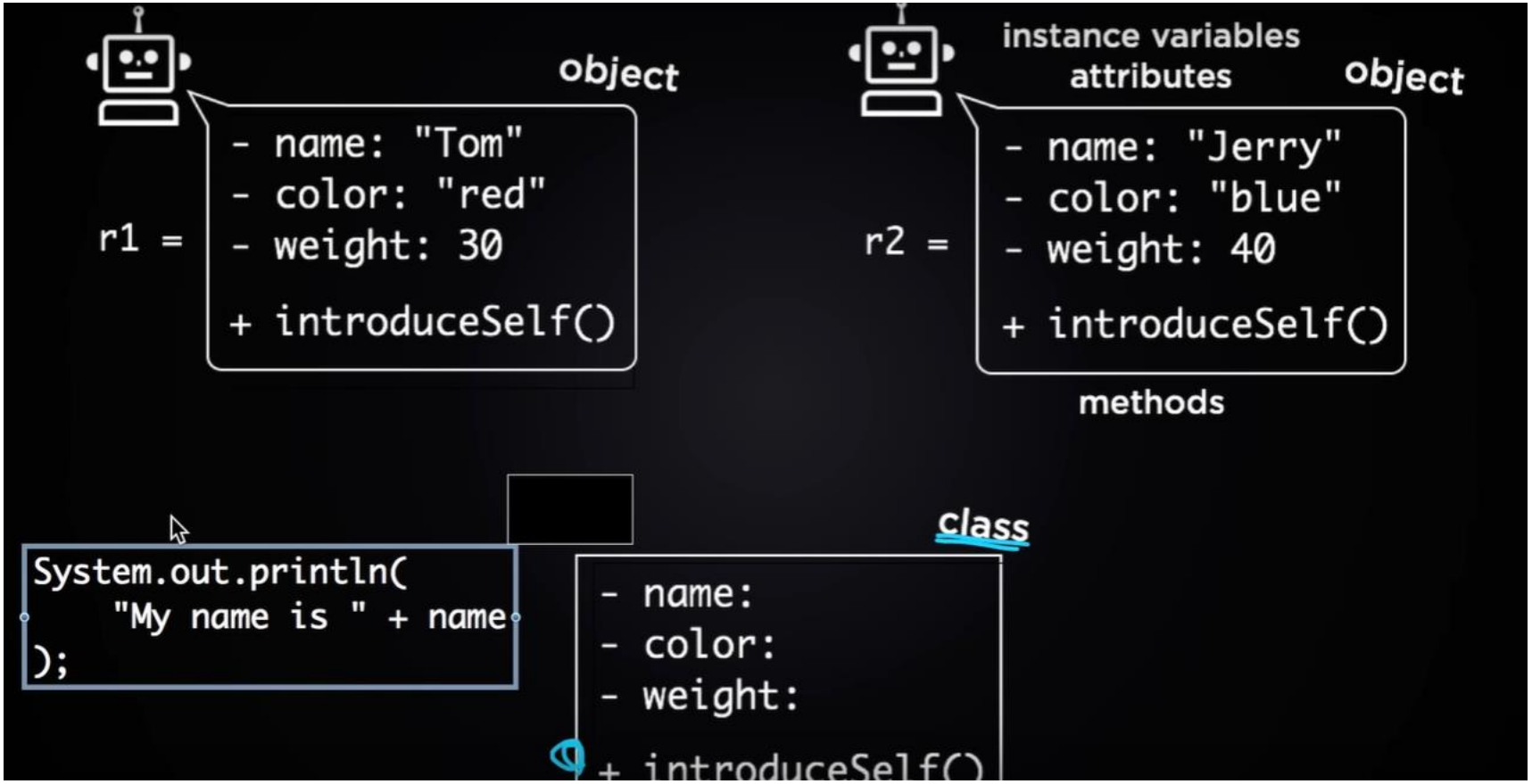

- name:
- color:
- weight:

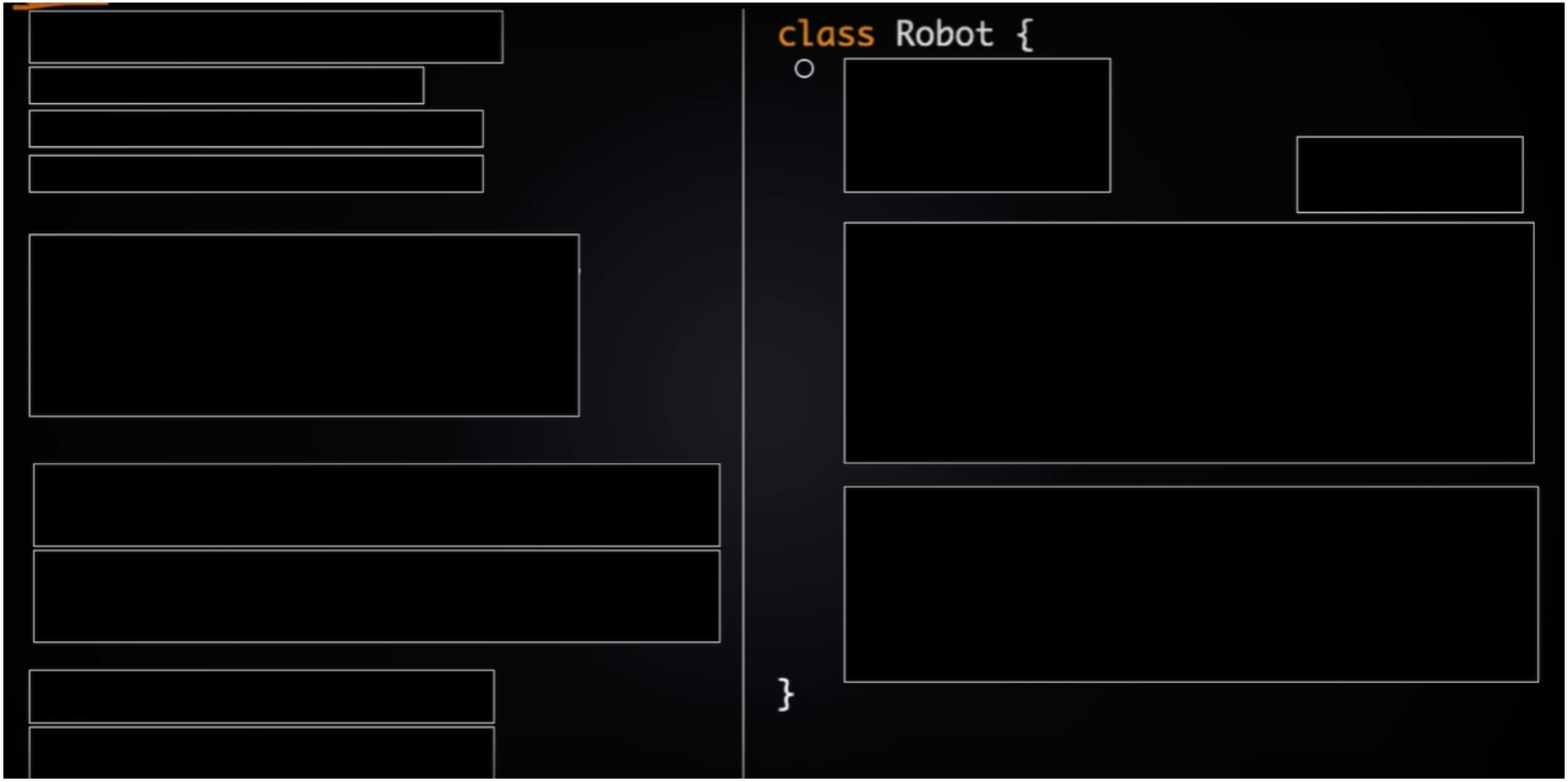

object
- name: "Tom"
- color: "red"
- weight: 30
r1 =


object
r1 =
- name: "Tom"
- color: "red"
- weight: 30
+ introduceSelf()


object
- name: "Tom"
- color: "red"
r1 =
- weight: 30
+ introduceSelf()
object
- name: "Jerry"
- color: "blue"
r2 =
- weight: 40
+ introduceSelf()


object
r1 =
- name: "Tom"
- color: "red"
- weight: 30
+ introduceSelf()
instance variables
attributes
object
r2 =
- name: "Jerry"
- color: "blue"
- weight: 40
+ introduceSelf()
methods
class
- name:
- color:
- weight:
+ introduceSelf()


object
- name: "Tom"
- color: "red"
- weight: 30
+ introduceSelf()
r1 =
instance variables
attributes
object
- name: "Jerry"
- color: "blue"
- weight: 40
+ introduceSelf()
r2 =
methods
System.out.println(
"My name is " + name
);
class
- name:
- color:
- weight:
+ introduceSelf()


class Robot {
}

```java
Robot r1 = new Robot();
r1.name = "Tom";
r1.color = "red";
r1.weight = 30;
```

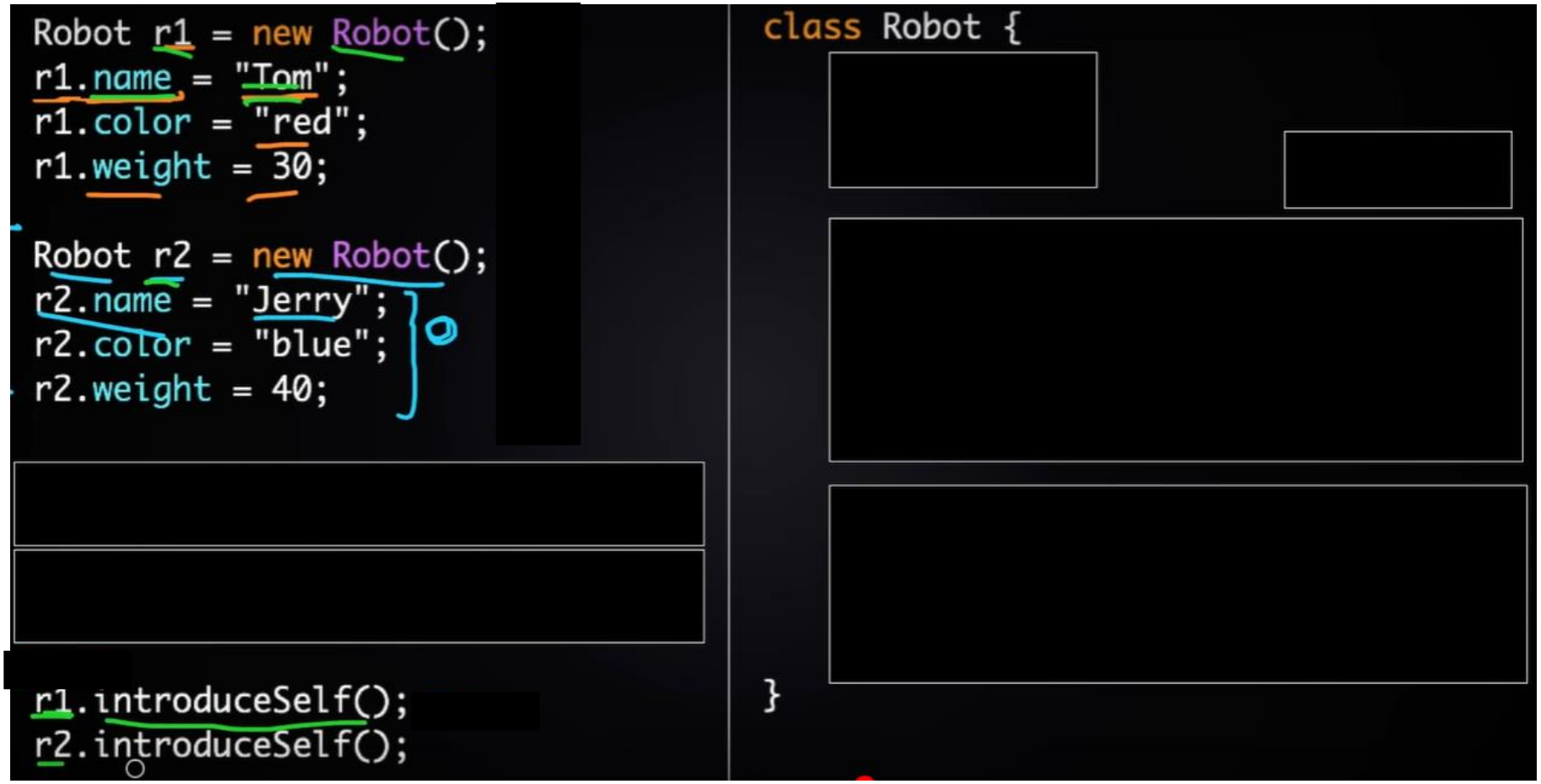
```java
class Robot {
}
```

```
Robot r1 = new Robot();
r1.name = "Tom";
r1.color = "red";
r1.weight = 30;

Robot r2 = new Robot();
r2.name = "Jerry";
r2.color = "blue";
r2.weight = 40;
```

```
class Robot {

}
```


Robot r1 = new Robot();
r1.name = "Tom";
r1.color = "red";
r1.weight = 30;
Robot r2 = new Robot();
r2.name = "Jerry";
r2.color = "blue";
r2.weight = 40;
r1.introduceSelf();
r2.introduceSelf();
class Robot {
}

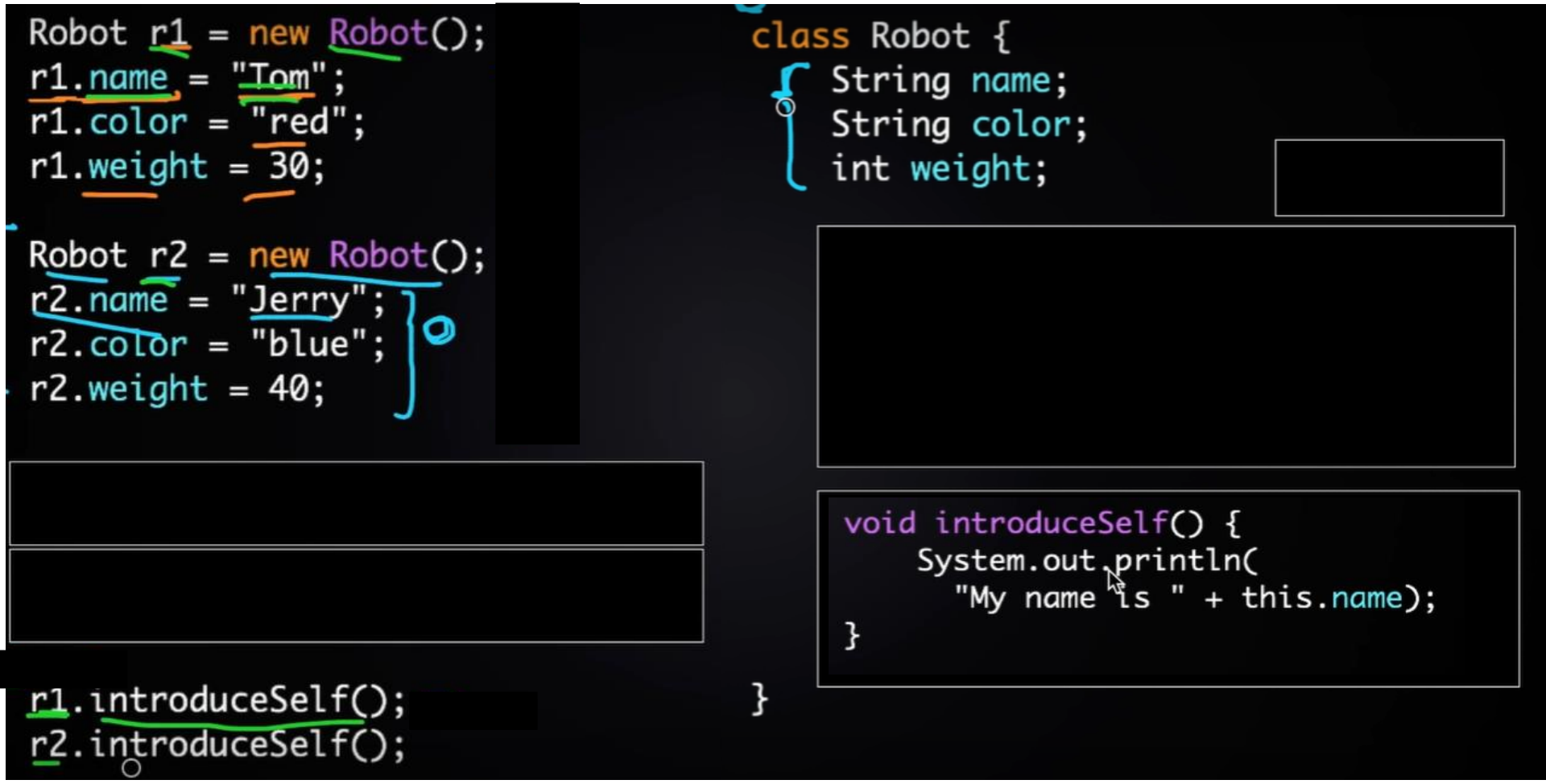
```java
Robot r1 = new Robot();
r1.name = "Tom";
r1.color = "red";
r1.weight = 30;

Robot r2 = new Robot();
r2.name = "Jerry";
r2.color = "blue";
r2.weight = 40;

r1.introduceSelf();
r2.introduceSelf();
```

```java
class Robot {
    String name;
    String color;
    int weight;

}
```


Robot r1 = new Robot();
r1.name = "Tom";
r1.color = "red";
r1.weight = 30;
Robot r2 = new Robot();
r2.name = "Jerry";
r2.color = "blue";
r2.weight = 40;
r1.introduceSelf();
r2.introduceSelf();
class Robot {
String name;
String color;
int weight;
void introduceSelf() {
System.out.println(
"My name is " + this.name);
}
}

```java
Robot r1 = new Robot();
r1.name = "Tom";
r1.color = "red";
r1.weight = 30;

Robot r2 = new Robot();
r2.name = "Jerry";
r2.color = "blue";
r2.weight = 40;

r1.introduceSelf();
r2.introduceSelf();
```

```java
class Robot {
    String name;
    String color;
    int weight;

    Robot(String n, String c, int w) {
        this.name = n;
        this.color = c;
        this.weight = w;
    }

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

constructor

```java
Robot r1 = new Robot();
r1.name = "Tom";
r1.color = "red";
r1.weight = 30;

Robot r2 = new Robot();
r2.name = "Jerry";
r2.color = "blue";
r2.weight = 40;

Robot r1 =
    new Robot("Tom", "red", 30);

r1.introduceSelf();
r2.introduceSelf();
```

```java
class Robot {
    String name;
    String color;
    int weight;

    Robot(String n, String c, int w) {
        this.name = n;
        this.color = c;
        this.weight = w;
    }

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

constructor

```java
Robot r1 = new Robot();
r1.name = "Tom";
r1.color = "red";
r1.weight = 30;

Robot r2 = new Robot();
r2.name = "Jerry";
r2.color = "blue";
r2.weight = 40;

Robot r1 =
    new Robot("Tom", "red", 30);
Robot r2 =
    new Robot("Jerry", "blue", 40);

r1.introduceSelf();
r2.introduceSelf();
```

```java
class Robot {
    String name;
    String color;
    int weight;

    Robot(String n, String c, int w) {
        this.name = n;
        this.color = c;
        this.weight = w;
    }

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

constructor

```java
Robot r1 = new Robot();
r1.name = "Tom";
r1.color = "red";
r1.weight = 30;

Robot r2 = new Robot();
r2.name = "Jerry";
r2.color = "blue";
r2.weight = 40;

Robot r1 =
    new Robot("Tom", "red", 30);
Robot r2 =
    new Robot("Jerry", "blue", 40);

r1.introduceSelf();
r2.introduceSelf();
```

```java
class Robot {
    String name;
    String color;
    int weight;

    Robot(String n, String c, int w) {
        this.name = n;
        this.color = c;
        this.weight = w;
    }

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

constructor

- https://www.youtube.com/watch?v=8yjkWGRlUmY