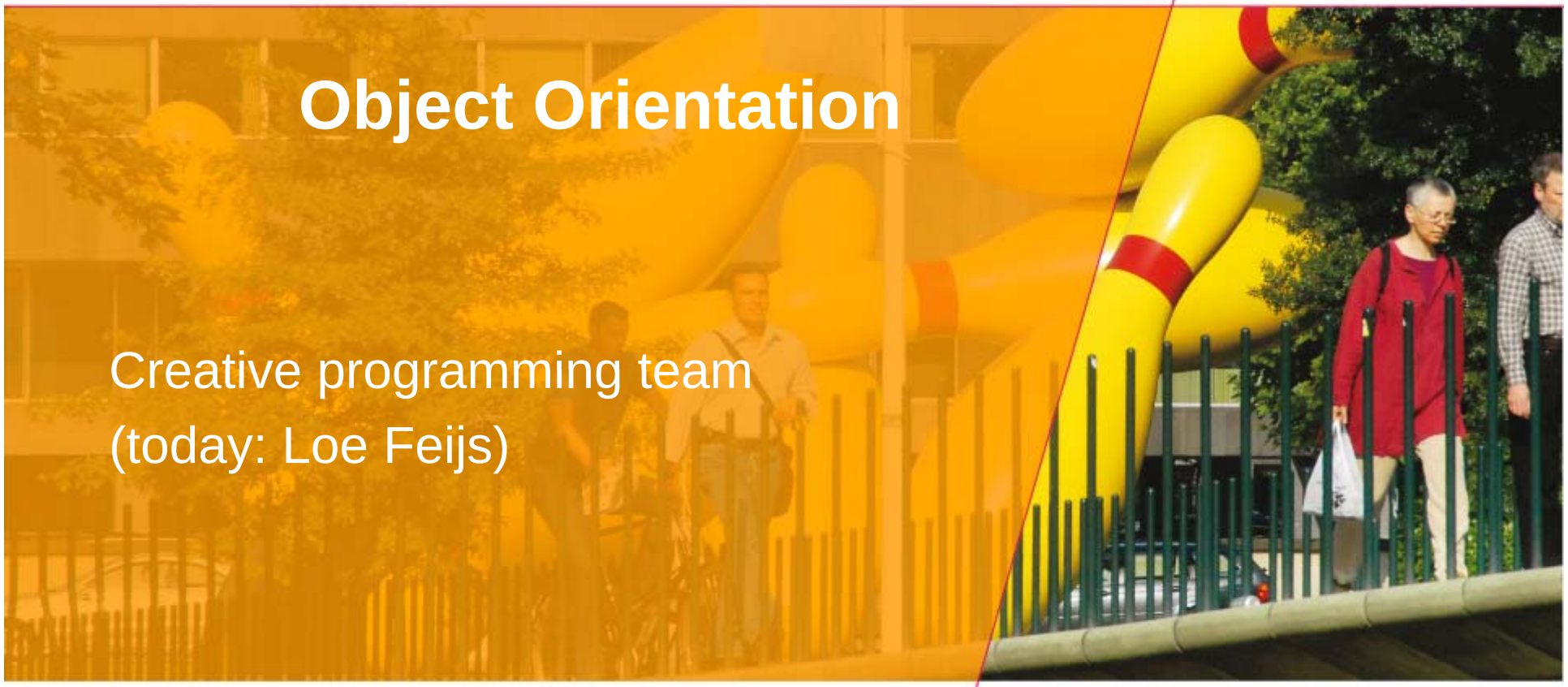
Object Orientation



Creative programming team

(today: Loe Feijs)

Technische Universiteit Eindhoven University of Technology

Designed Intelligence Group

Where innovation starts

Content

- Background
- Encapsulation
- Planet earth example
- Private and public
- EPD example
- Super- and subclass
- Cars example
- Bouncing balls revisited

Background

Object Oriented Programming

- a revolutionary extension of programming
- extends earlier programming abstractions
- is the leading programming paradigm
- similar to techniques of thinking about problems in other domains e.g. architecture

Background

Program consist of many “things” (objects)

- there are different kinds of “things”
- objects are created as instances of *classes*
- objects can have an internal state and *components*.
- objects exchange *messages*
- if object *A* sends message to *B* then *B* does something (and returns a result to *A*)
- results can be **`boolean`**, **`int`**, **`float`** or **`string`**
- or they can be an object themselves or there is no result (**`void`**).
- there is some main object with a loop that starts everything off

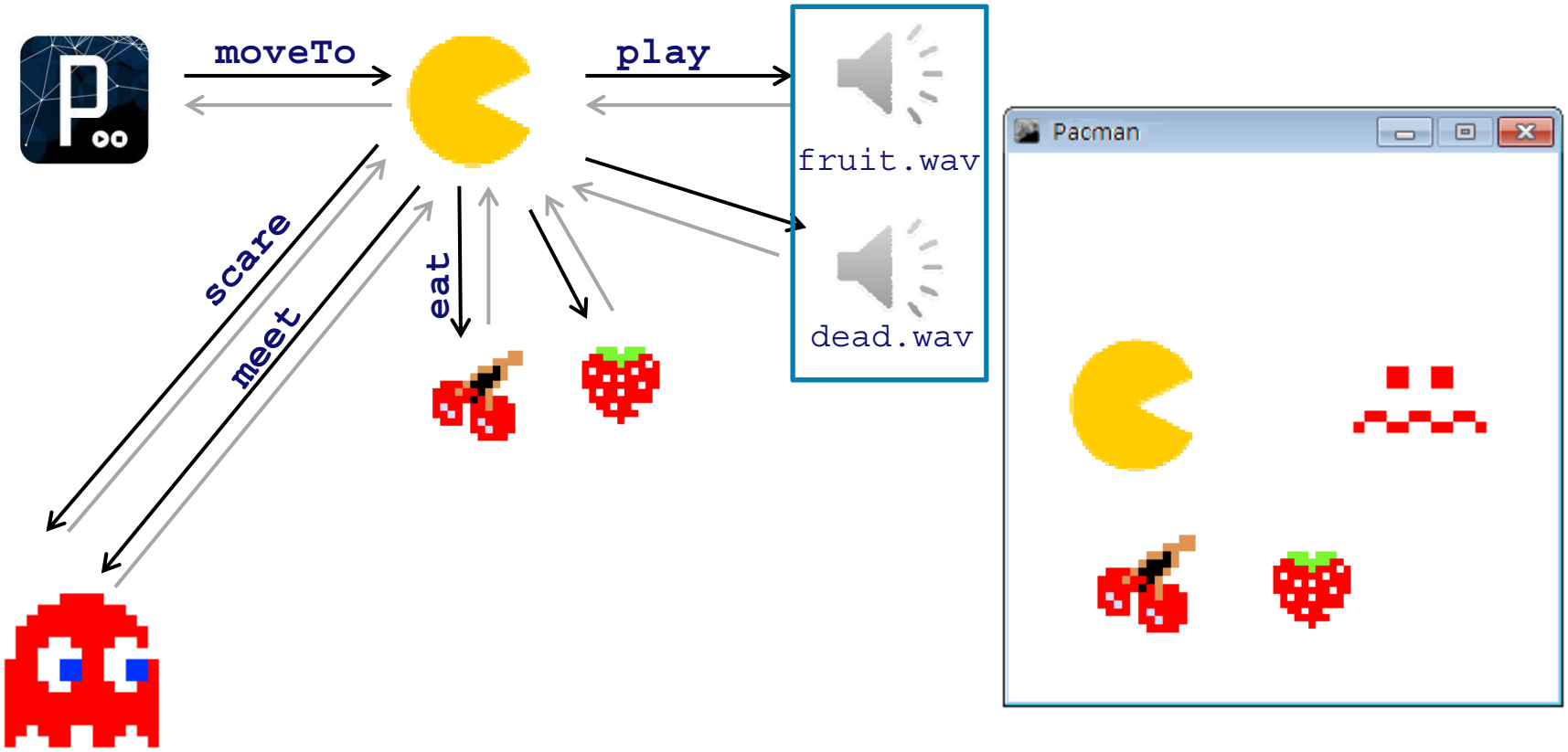
Background

- objects encapsulate state as a collection of instance variables
- objects encapsulate behaviour via methods invoked by messages

```
Rectangle
---------------------
int side1;
int side2;
---------------------
cirumference
area
moveTo
```

Background

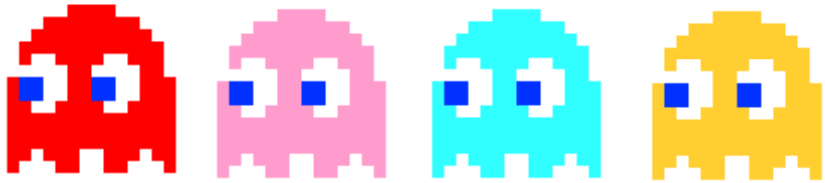
Program: a world of objects



Original design by Toru Iwatani, 1977, © Namco

Encapsulation

```
Pacman me;
AudioPlayer dead, fruit;
Ghost blinky, inky, pinky, clyde;

me = new Pacman();
blinky = new Ghost(#FF0000);
inky   = new Ghost(#0000FF);
pinky  = new Ghost(#FFAAAA);
clyde  = new Ghost(#FFAA00);

player = new Minim(this);
dead   = player.loadFile("dead.wav");
fruit  = player.loadFile("fruit.wav");

me.moveTo(1);
if (me.meet(inky)) {
    inky.die();
    dead.play();
}
```

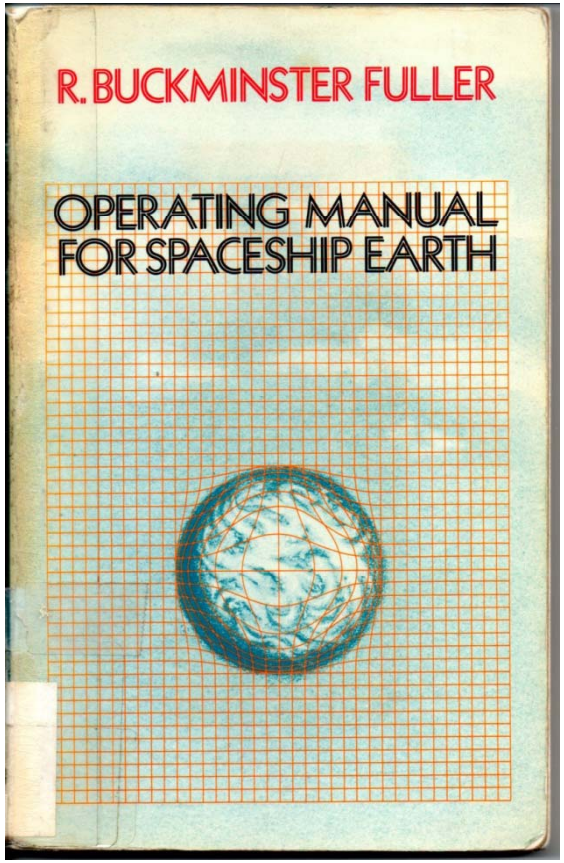


Encapsulation

Encapsulation is a technique for

- creating for objects with encapsulated state
- and encapsulated behaviour
- hiding implementation details
- protecting the state information of objects
- putting objects in control
- facilitating modularity, code reuse and maintenance

Example: planet earth



Encapsulation

```
//planet earth
  float diameter;
  color surface;
  float angle;

void setup(){
  size(800, 600);

//initialize planet earth
  diameter = 12756;     //in 1000km
  surface = #00FF00;    //it's a green planet
  angle = random(350); //it's a new day
}
```

Encapsulation

```
void orbit(float radiusX, float radiusY, float orbitTime){

    // draw ellipse first, so it's under planet earth
    drawOrbit(radiusX, radiusY);
    float px = cos(radians(angle))*radiusX/2;
    float py = sin(radians(angle))*radiusY/2;

    // draw planet earth next
    fill(surface);
    noStroke();
    ellipse(px, py, diameter/2000, diameter/2000);

    //move forward
    angle += (1/orbitTime);
}
```

Encapsulation

```
void drawOrbit(float radiusX, float radiusY){

     //make the entire orbit visible
     stroke(255,50);
     float angle=0;
     for (int i=0; i<360; i++){
         point(cos(radians(angle))*radiusX/2,
               sin(radians(angle++))*radiusY/2);
     }
}
void draw(){
     background(0);
     translate(width/2, height/2);
     orbit(150, 153, 1.00);
     //min and max distance to sun in 1000000km
     //rotation time in years
}
```

Encapsulation

Now we want not *one* planet but a lot of them

- class concept
- declare variables just like before
- functions are now called “methods”
- and can be invoked by messages using “.” notation

Encapsulation

```
class Planet{
  float diameter;
  color surface;
  float angle;

  Planet(float diameter, color  surface){
        this.angle = random(350);
        this.diameter = diameter;
        this.surface = surface;
  }
```

Encapsulation

```
public void orbit(float radiusX, float radiusY, float orbitTime){

    // draw ellipse first, so it's under the planet
    drawOrbit(radiusX, radiusY);
    float px = cos(radians(angle))*radiusX/2;
    float py = sin(radians(angle))*radiusY/2;

    // draw planet next
    fill(surface);
    noStroke();
    ellipse(px, py, diameter/2000, diameter/2000);

    //big planets get a red dot award
    if (diameter > 100000){
        fill(#FF5522);
        ellipse(px, py+20, 10, 10);
    }

    //move forward
    angle += (1/orbitTime);
  }
```

Encapsulation

```
void drawOrbit(float radiusX, float radiusY){

        //make the entire orbit visible
        stroke(255, 50);
        float angle=0;
              for (int i=0; i<360; i++){
                   point(cos(radians(angle))*radiusX/2,
                         sin(radians(angle++))*radiusY/2);
              }
        }
}
```

Encapsulation

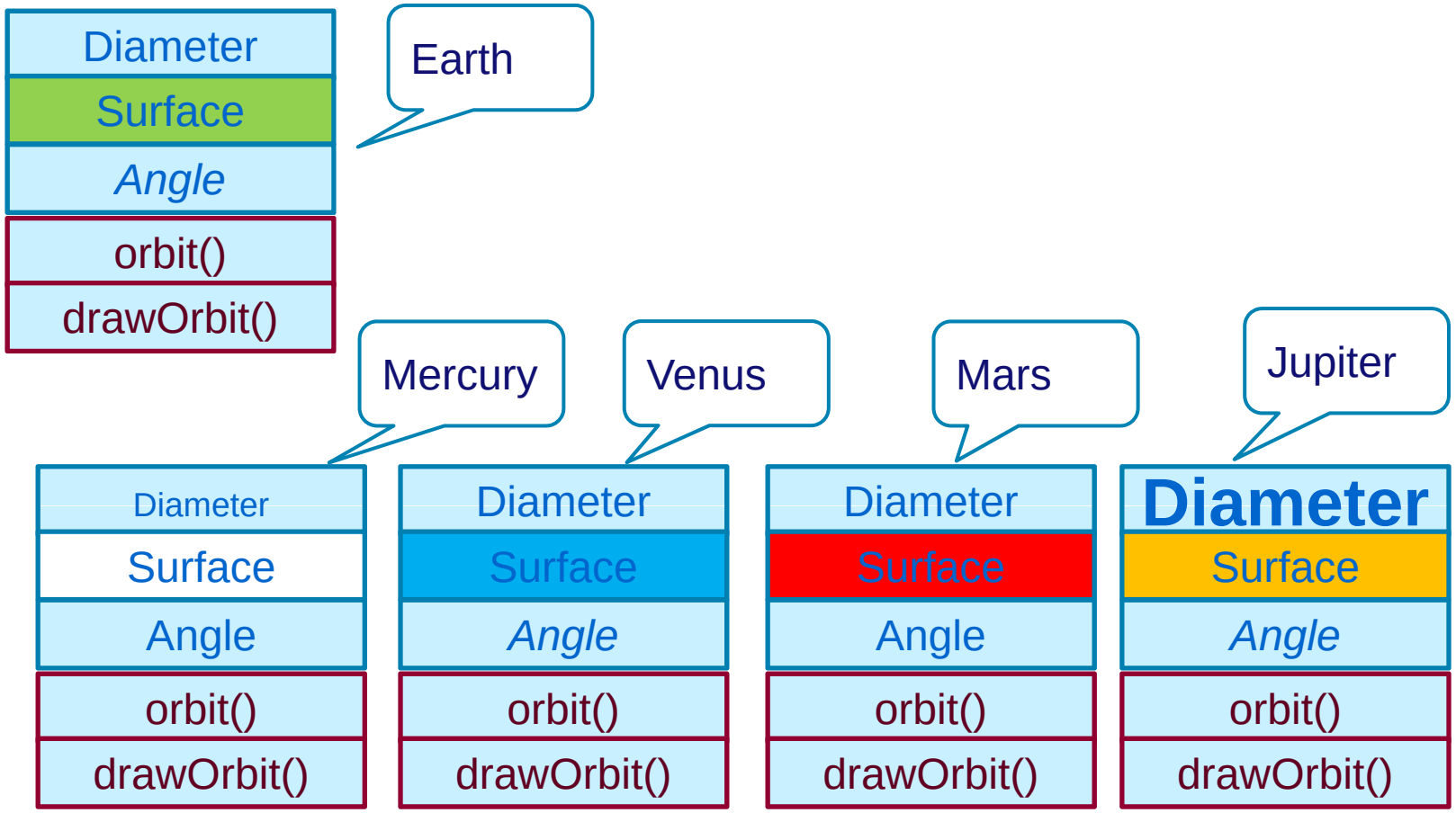
How do I create an object ?

- each class has a constructor with the same name
- `Planet myfirstPlanet = new Planet();`
- the variable `myfirstPlanet` is assigned a reference to the new `Planet` object
- uses the first constructor, there may be more complex constructors ..
- `mercury = new Planet(4878, #DDDDDD);`

Encapsulation

```
//source: http://processing.org/discourse/beta/num_1162399456.html
//posted by ira, (in reply to: Building a graphic of a solar system)
//adapted by (c) 2012 Loe Feijs TU/e

Planet mercury;
Planet venus;
Planet earth;
Planet mars;
Planet jupiter;

void setup(){
     size(800, 600);//,P3D);
     mercury = new Planet(4878, #DDDDDD);
     venus = new Planet(12104, #6688FF);
     earth = new Planet(12756, #00FF00);
     mars = new Planet(6788, #FF0000);
     jupiter = new Planet(142796, #FFEE33);
}
```

Encapsulation



Encapsulation

```
void draw(){
     background(0);
     translate(width/2, height/2);

     mercury.orbit(58, 59, 0.24); //distances in 1000000km
     venus.orbit(108, 109, 0.62); //rotation time in years
     earth.orbit(150, 153, 1.00);
     mars.orbit(230, 232, 1.88);
     jupiter.orbit(778, 782, 11.87);
}
```

Encapsulation

Encapsulation

What we achieved

- every object has it's own memory, which consists of variables and other objects
- each object is like a miniature computer itself
- its like a specialized processor performing a specific task
- every object is an instance of a class. A class groups similar objects
- the class is the repository for behaviour associated with an object
- behaviour is associated with classes, not with individual instances.
- all objects of a given class use the same method in response to similar messages

Recap

- Background
- Encapsulation
- Planet earth example
- Private and public
- EPD example
- Super- and subclass
- Cars example
- Bouncing balls revisited

Public and private

Towards even better encapsulation

- prevent that anyone accesses every variable
- prevent that anyone can invoke every method
- keep the fields of an object inside the class
- distinguish with attributes “public” or “private”

Public and private

Public view:

- those features (data or behaviour) that other objects can see and use

Private view:

- those features (data or behaviour) that are only used within the object.

Note:

- in java or processing keywords *public* and *private* are applied individually

to every component or method

Example: EPD

Oud-voorzitter LHV: Maak

medischcontact.artsennet.nl/nieuws-26/nieuwsbericht/124925/oudvoorzitter-lhv-maak-patient-eigenaar-epd.htm

Medisch Contact

en huiselijk geweld Kindermishandeling en volwassenengeweld knmg www.knmg.nl/meldcode

Artsennet en deelnemers | Inloggen | Registreren Typ uw zoekopdracht zoek

Home Nieuws Video Rubrieken Dossiers Tuchtrecht Kennis Vacaturebank Tests Webshop Info Adverteren

U bent nu hier: Nieuws | Nieuwsbericht

Oud-voorzitter LHV: Maak patiënt eigenaar epd

Publicatie	05 december 2012
Jaargang	2012
Rubriek	NieuwsReflex
Auteur	Simone Paauw

De patiënt moet eerstverantwoordelijke worden over het eigen medisch dossier. Dat stelt oud-LHV-voorzitter H. Bessem (van 1978 tot 1983) in een brief aan minister Schippers.

Beeld: Thinkstock

Bessem vindt dat het de hoogste tijd is dat iedere volwassen Nederlander zeggenschap krijgt over zijn eigen medisch dossier en niet meer afhankelijk is van koepelorganisaties LHV, ZN en NPCF en andere organisaties in de gezondheidszorg. Bessem stelt dat veel huisartsen en patiënten verontrust zijn, omdat die organisaties buiten het parlement om toch de uitwisseling van medische gegevens mogelijk willen maken door de instelling van een landelijk schakelpunt.

Tweets

#medischcontact

Karin Timm Net gelezen in @MedZineApp: 'Rapport over ggz had duidelijker moeten zijn' bit.ly/10Su53A
2 days ago · reply · retweet · favorite

riannerst "Met koffie en @medischcontact op de bank" toegevoegd aan zaterdag-lijstje. En dat is nu aan de beurt! Dochter met haar boek naast me. #fb
yesterday · reply · retweet · favorite

jharbers Zonder investering in antibiotica beleid straks 50% postoperatieve infecties

Join the conversation

source: medisch contact, retrieved 18-3-2013

Example: EPD

```
class Doctor {
      String clinic;
      Doctor(String c){
             clinic = c;
      }
}

class Test {
      boolean valid;
      boolean result;
      Test(boolean v, boolean r) {
           valid = v;
           result = r;
      }
      String publish(){
             if (valid)
                if (result) return "true"; else return "false";
             else return "INVALID";
      }
}
```

Example: EPD

```
class Citizen {
      String birthDate;
      private String passWord;
      private String[] conditions;
      private Doctor[] trusted;
      private int nrConditions;
      private int nrTrusted;

      Citizen (String pwd,String date){
              passWord = pwd;
              birthDate = date;
              nrConditions = 0;
              conditions = new String[1000];
              nrTrusted = 0;
              trusted = new Doctor[1000];
      }
```

Example: EPD

```
public void addCondition(Doctor dr, String condition){
        if (trust(dr))
            conditions[nrConditions++] = condition;
}
private boolean hasCondition(String condition){
        boolean has = false;
        for (int i=0; i< nrConditions; i++)
             if (conditions[i].equals(condition))
                 has = true;
        return has;
}
public void addDoctor(String pwd, Doctor dr){
        if (pwd.equals(passWord))
           trusted[nrTrusted++] = dr;
}
```

Example: EPD

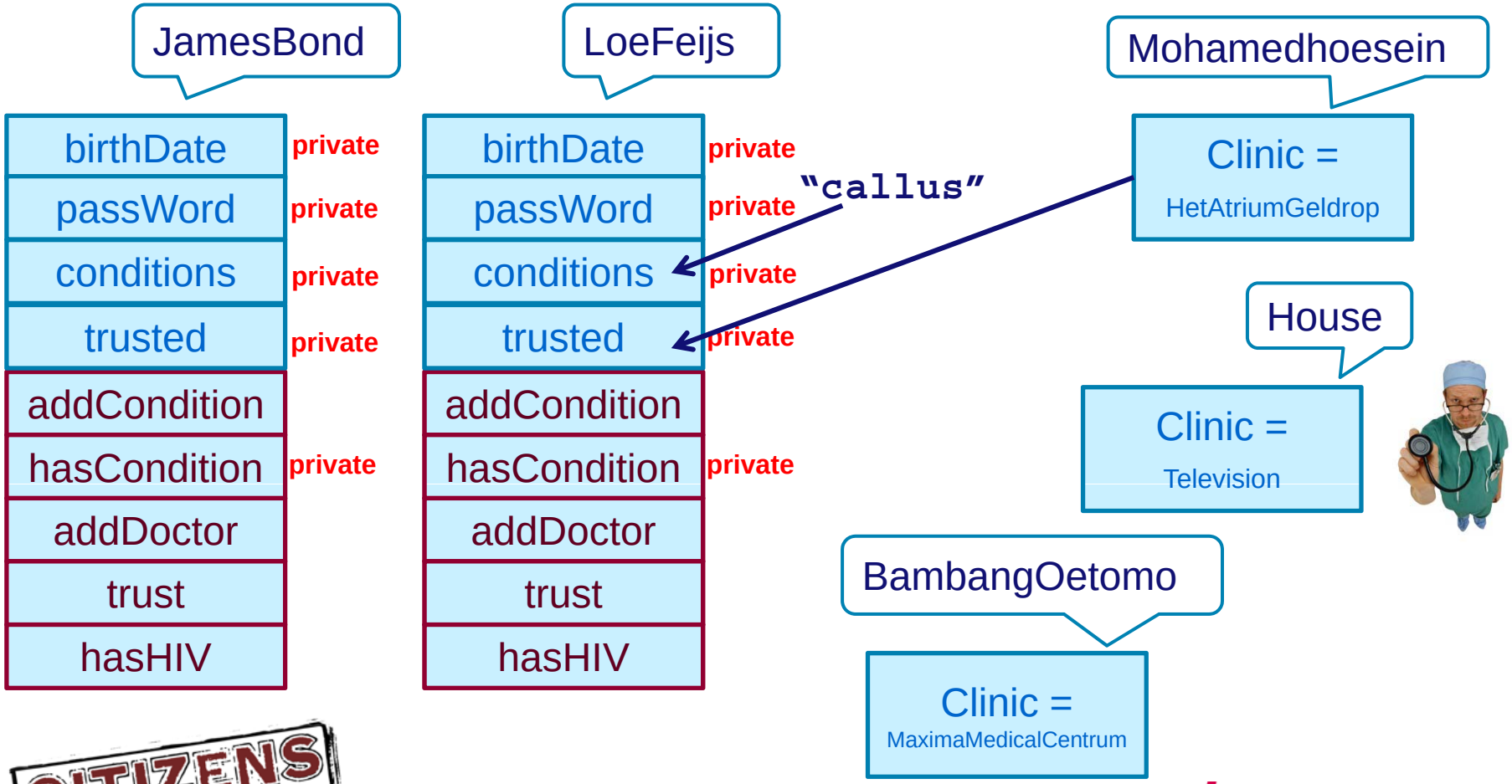
```
        public boolean trust(Doctor dr){
                boolean found = false;
                for (int i=0; i< nrTrusted; i++)
                        if (trusted[i] == dr)
                                found = true;
                return found;
        }

        public Test testHIV(Doctor dr){
                Test tst = new Test(trust(dr),false);
                if (tst.valid)
                        for (int i=0; i<nrConditions; i++)
                                if (conditions[i].equals("HIV"))
                                        tst.result = true;
                return tst;
        }
}       //end class Citizen
```

Public and private

What we achieved

- the fields **`passWord`**, **`conditions`**, and **`trusted`** are private
- they can only be accessed by **`Citizen`** objects
- or indirectly by invoking methods of the class **`Citizen`**
- the actions of **`addCondition`** and **`testHIV`** include a check (doctor trusted?)

Public and private



source: www.meteor17.com

Public and private

```
Citizen LoeFeijs = new Citizen("tokipona","04081954");
Citizen JamesBond = new Citizen("skyfall","03071953");

Doctor House = new Doctor("Television");
Doctor Mohamedhoesein = new Doctor("HetAtriumGeldrop");
Doctor BambangOetomo = new Doctor("MaximaMedicalCentrum");

void setup(){
     LoeFeijs.addDoctor("tokipona",Mohamedhoesein);
     LoeFeijs.addCondition(Mohamedhoesein,"callus");
     LoeFeijs.addCondition(House,"lupus");
     Test t = LoeFeijs.testHIV(Mohamedhoesein);
     println(t.publish());
}
```

```
false
```

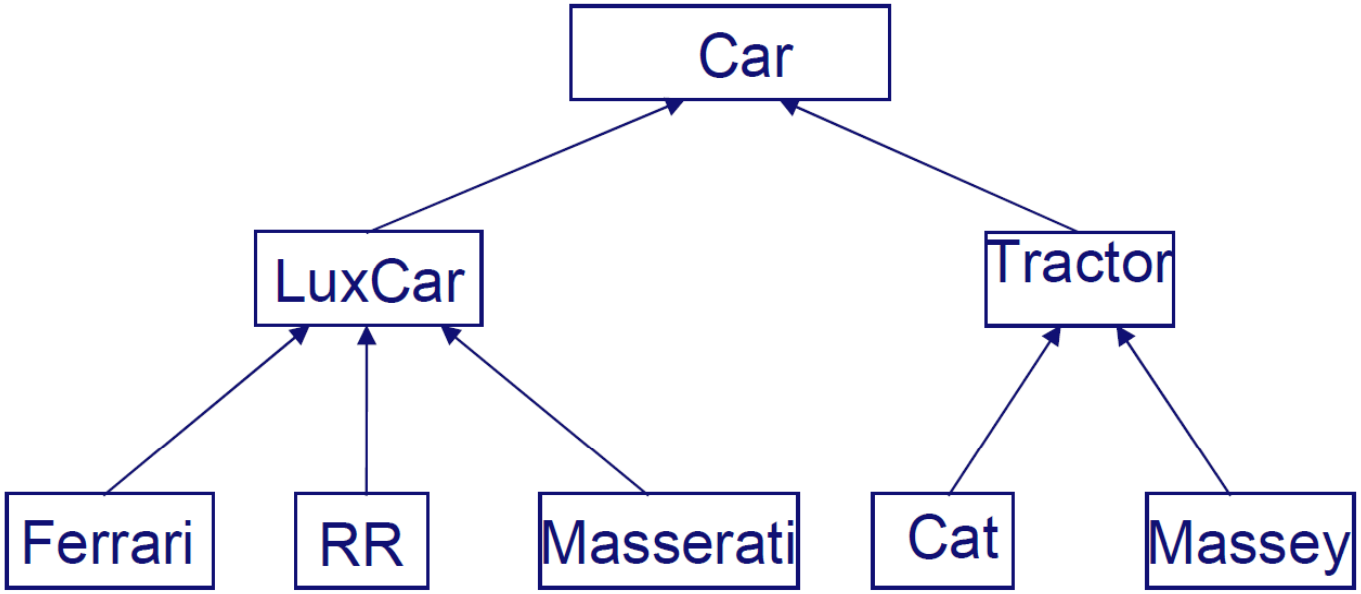
Recap

- Background
- Encapsulation
- Planet earth example
- Private and public
- EPD example
- Super- and subclass
- Cars example
- Bouncing balls revisited

Super- and subclass

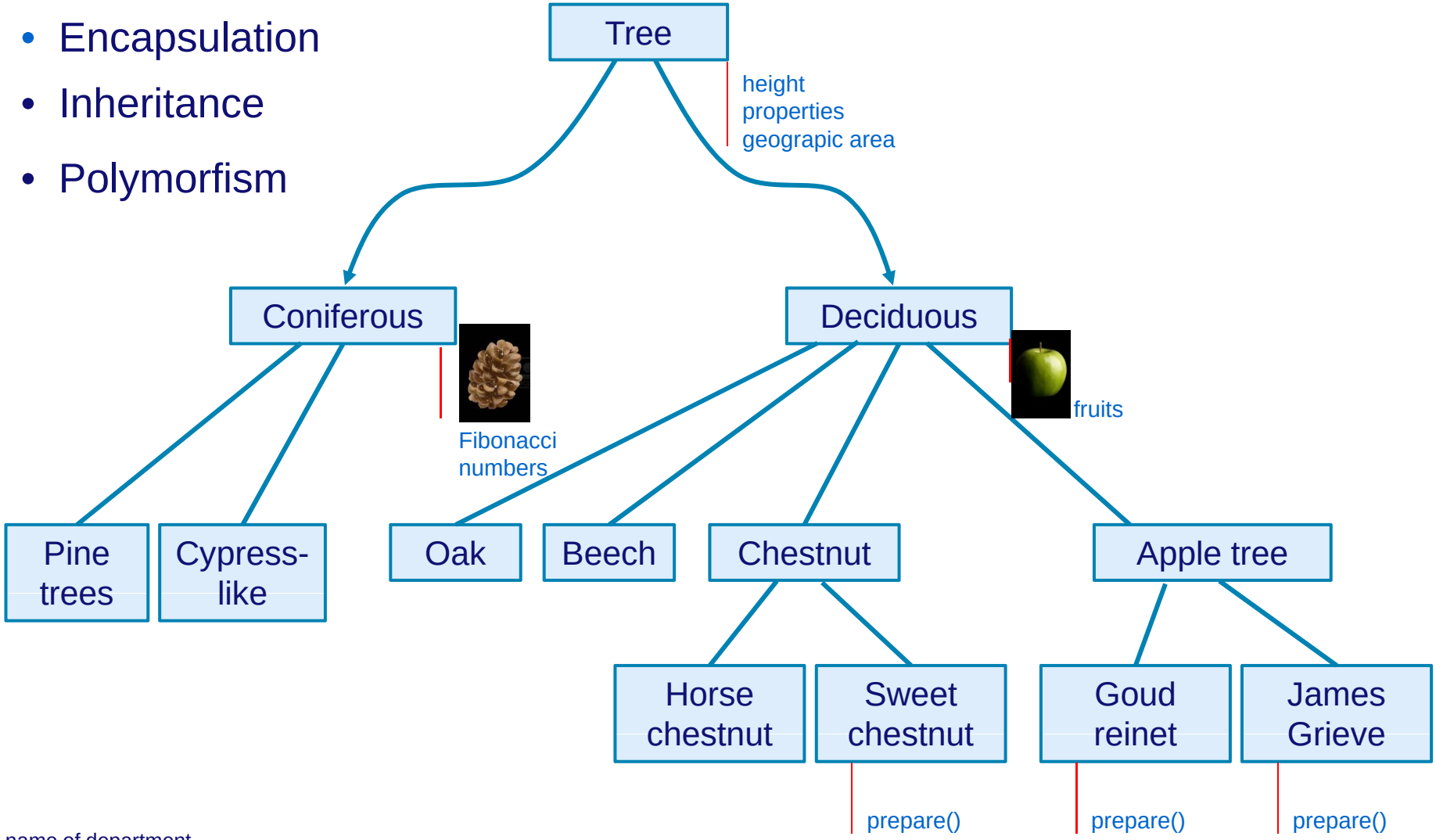
Classes form a hierarchy

- superclass is the parent and subclass is a child
- subclasses “extend” (i.e. specialize) their superclass



Super- and subclass (ideas from biology)

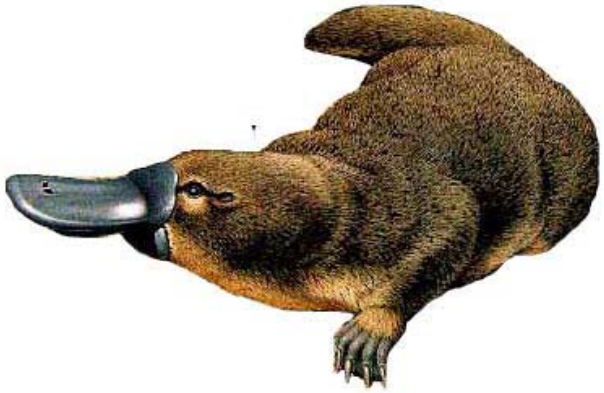
- Encapsulation
- Inheritance
- Polymorfism



Super- and subclass

Subclasses can alter or override inherited information

- all mammals give birth to living young
- all fish have gills



Source platypus image:
http://www.vakantieweblog.eu/vakanties/australie2008/verslag/vogelbekdier.jpg

Generic class definition

```
class ClassName extends OtherClass {
        //properties or components
        int property1;
        float property2;
        rectangle component3;

        //constructors
        ClassName(){}
        ClassName(int prop1,float prop2){
                property1 = prop1;
                property2 = prop2;
        }
```

Generic class definition, continued

```
        //methods
        void setProperty1(int prop1){
             property1 = prop1;
        }
        int getProperty1(){
            return property1;
        }
        …
        other ... specific methods

} //class ends
```

Inheritance in Java

A note on inheritance in Java:

- a single root class: `Object`
- all classes inherit from some class, default `Object`

Example: cars

Scooped by **Good Things From Italy** onto **Good Things From Italy - Le Cose Buone d'Italia**

De mooiste Ferrari's van Sergio Pininfarina

Rescoop

From **www.autoitalia.nl** - November 5, 3:28 PM

Halverwege dit jaar overleed ontwerper Sergio Pininfarina, een naam die iedereen die ook maar iéts met auto's heeft ongetwijfeld zal kennen. Naast de nodige betaalbare auto's ontwierp Pininfarina vooral veel exclusieve auto's waarbij het merk met het steigerende paard een groot deel hiervan voor zijn rekening nam.

Meer dan 100 verschillende modellen kwamen er uit de pen van de ontwerper en dat vindt Ferrari reden genoeg voor een fraaie expositie van de fraaiste modellen. 22 exemplaren zijn er uitgestald in het Ferrari museum in Maranello voor de expositie die luistert naar de naam 'The Great Ferraris of Sergio Pininfarina Exhibition'.

Example: class car

```
class Car {
  color carCol;
  int xPos, yPos;
  int tireWidth = 13;

  Car() {
    carCol = color(0, 0, 0);//as Ford said
    xPos = 123;
    yPos = 134;
  }

  void setyPos( int y) {
       yPos = y ;
  }
  void carPaint(color desiredColor) {
       carCol = desiredColor;
  }
```

Source: wallpapersup.net/ford-mustang-v6-black/

Example: class car

```
  void DrawCar() {
        //body
        fill(carCol)  ;
        ellipse(xPos, yPos, 120, 20);
        //front tires
        fill(0);
        rect(xPos+20, yPos+10, 20, tireWidth);
        rect(xPos+20, yPos-10-tireWidth, 20, tireWidth);
        //rear tires
        rect(xPos-40, yPos+10, 20, tireWidth);
        rect(xPos-40, yPos-10-tireWidth, 20, tireWidth);
  }

  void drive(int s) {
        xPos += s;
        xPos = xPos % width;
        yPos = yPos % height;
  }
} //end class
```

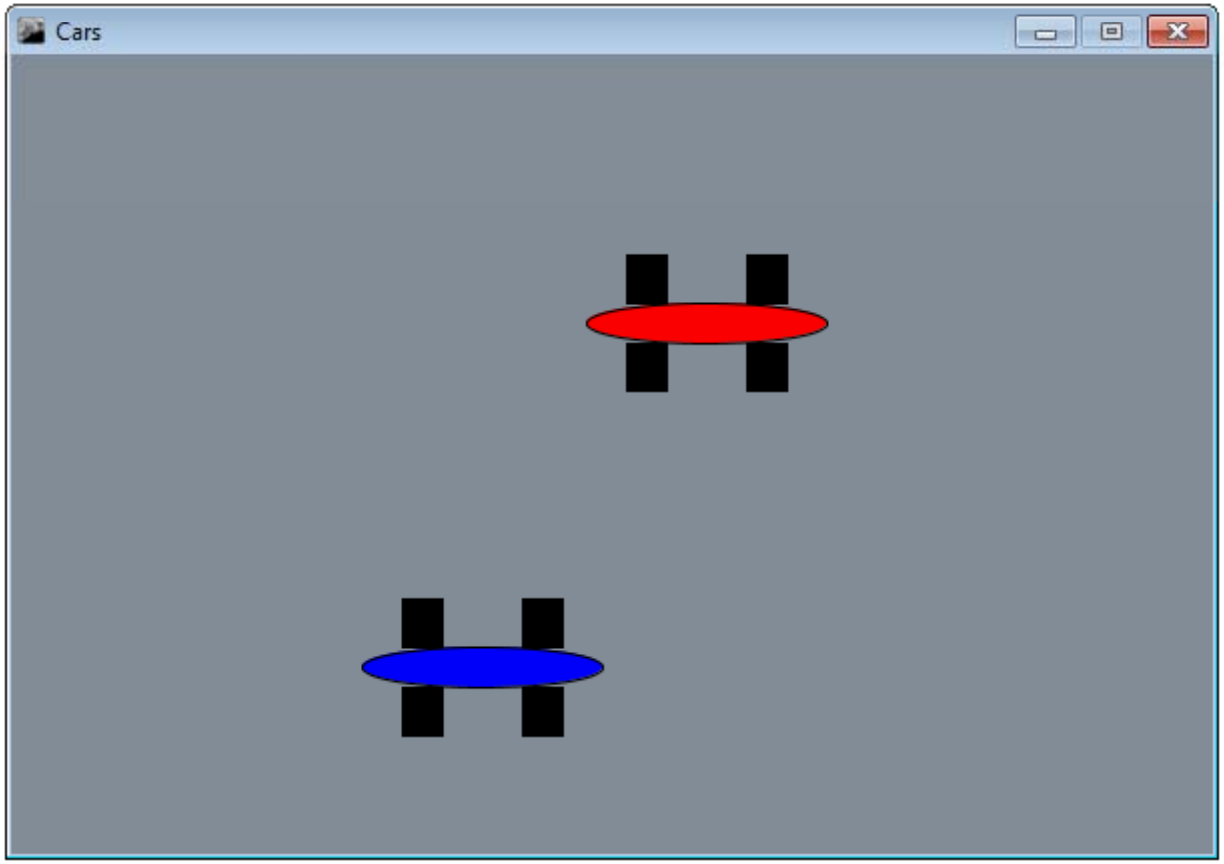
Example: class car

```
class LuxCar extends Car {

  LuxCar() {
     tireWidth = 24;
  }
}

class CrossCar extends LuxCar {
   int cnt = 0;

   void drive(int s) {
        cnt++;
        cnt%=20;
        if (cnt<10) yPos++; else yPos--;
        xPos += s;
        xPos = xPos % width;
        yPos = yPos % height;
   }
}
```

Example: class car

```
Car mycar;
Car yourcar;

void setup() {
     size(600, 400);
     mycar = new LuxCar();
     yourcar = new CrossCar();
     yourcar.setyPos(300);
     mycar.carPaint( color(250, 0, 0) );
     yourcar.carPaint( color(0, 0, 250) );
}

void draw()  {
     background(130,140,150);
     mycar.DrawCar();
     mycar.drive(2);
     yourcar.DrawCar();
     yourcar.drive(1);
};
```

Recap



Encapsulation

What we achieved

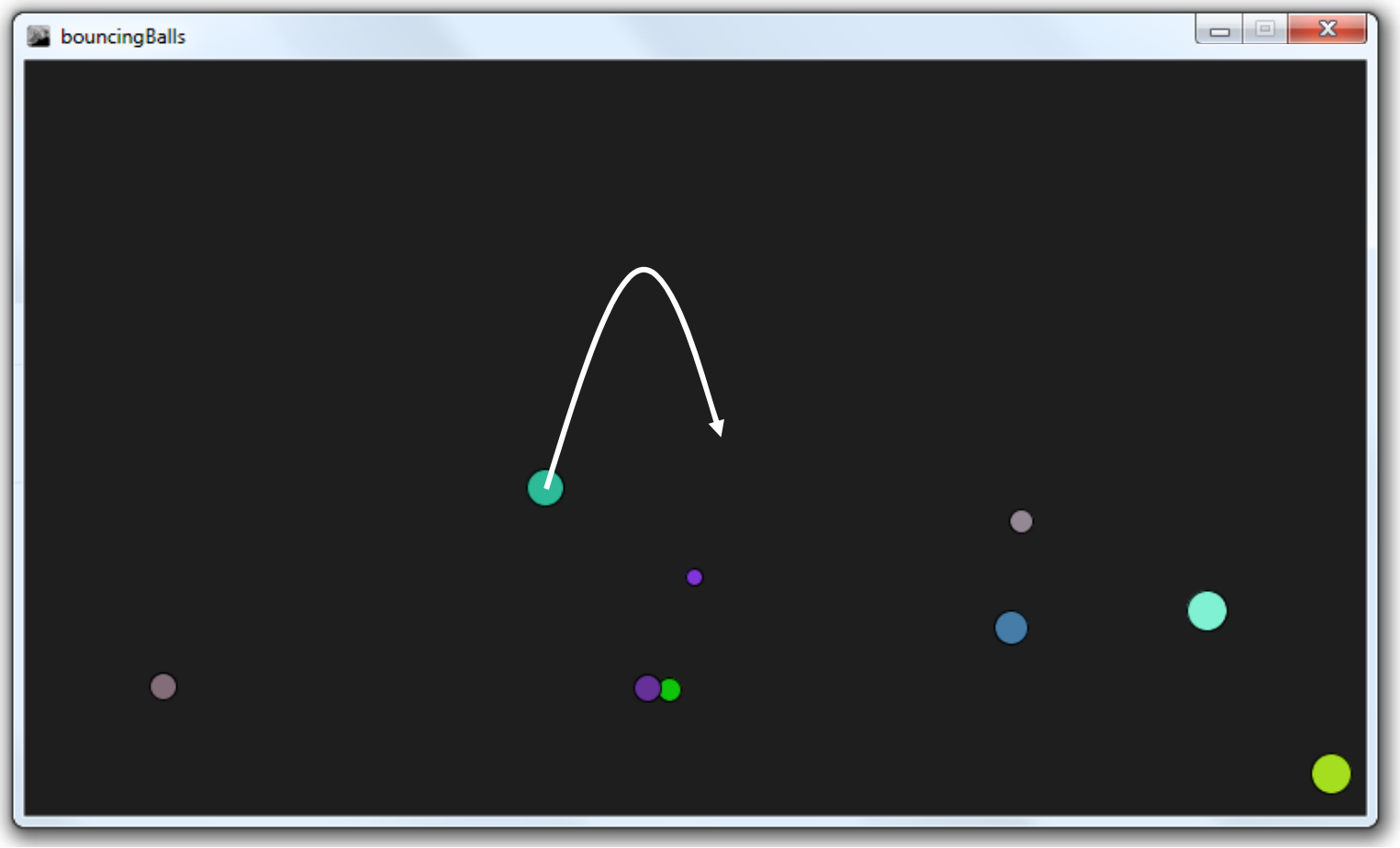
- several types of `Car` objects
- easy creation of new `Car` objects of these types
- common properties and methods modeled only once
- `LuxCar` has a different default value for property tireWidth
- `CrossCar` has one additional internal counter variable `cnt`
- `CrossCar` has different `drive` behaviour, overriding the original

Another example: the bouncing balls revisited

- Collision detection
  - if not many objects: quadratic testing
  - otherwise: use smart bounding boxes

- Physical simulation
  - Regular update of object parameters
  - First speed, then position

Collision detection & physical simulation



```
//OLD STYLE: ONE ARRAY FOR EACH PARAMETER

int N = 10;//how many balls

float[] posX = new float[N];
float[] posY = new float[N];
float[] dirX = new float[N];
float[] dirY = new float[N];
float[] radi = new float[N];
float[] mass = new float[N];

int R[] = new int[N];//colors
int G[] = new int[N];
int B[] = new int[N];
```

New style: balls as objects

```
class Ball {
  float posX;
  float posY;
  float dirX;
  float dirY;
  float radi;
  float mass;
  int R;
  int G;
  int B;

  //constructor
  Ball()//todo

  //actions
  void newPosition() //todo
  void newVelocicity() //todo
  private boolean collision(Ball other){
}
```

New style: balls as objects

```
//constructor method

Ball (float ix, float iy,
      float dx, float dy,
      int r, int iR, int iG, int iB){

      posX = ix;
      posY = iy;
      dirX = dx;
      dirY = dy;
      radi = r;
      mass = 3.14159 * r * r;
      R=iR;
      G=iG;
      B=iB;
}
```

New style: balls as objects

```
//action

void newPosition(){
     posX += dirX;
     posY += dirY;

     fill (R, G, B);
     ellipse (posX, posY, 2*radi, 2*radi);

     if (posX >= width) {dirX = -dirX;}
     if (posX <= 0) {dirX = -dirX;}

     if (posY > height - radi) {
         dirY = -abs(dirY);
         if (abs(dirY) < 1) {
             dirY = 0;
         }
         posY = height - radi;
     }
}
```

Bounce at right wall

Bounce at left wall

Bounce at floor

New style: balls as objects

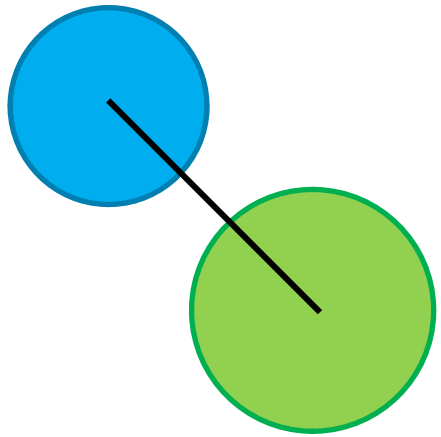
```
//action

void newPosition(){
        posX += dirX;
        posY += dirY;

        fill (R, G, B);
        ellipse (posX, posY, 2*radi, 2*radi);

        if (posX >= width) {dirX = -dirX;}
        if (posX <= 0) {dirX = -dirX;}

        if (posY > height - radi) {
            dirY = -abs(dirY);
            if (abs(dirY) < 1) {
                dirY = 0;
            }
            posY = height - radi;
        }
    }
```

Bounce at right wall

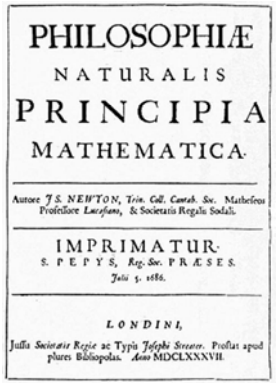
Bounce at left wall

Bounce at floor

```
//action continued

 private boolean collision(Ball other){

          float dd, rr;

          dd = dist(posX, posY, other.posX, other.posY);
          rr = radi + other.radi;

          if  (this != other && dd < rr)
               return true;
          else return false;
  }
```



```
//action continued

 void newVelocity() {

      for (int i = 0; i < N; i++) {
           Ball other = Balls[i];
           if (this.collision(other)){

                //Calculate new velocity
                dirX = //big formula from physics book
                dirY = //idem
      }
      dirY = dirY + gravity;
```


PHILOSOPHIÆ
NATURALIS
PRINCIPIA
MATHEMATICA.

Autore J S. NEWTON, Trin. Coll. Cantab. Soc. Matheseos Professore Lucasiano, & Societatis Regalis Sodali.

IMPRIMATUR.
S. PEPYS, Reg. Soc. PRÆSES.
Julii 5. 1686.

LONDINI,
Jussu Societatis Regiæ ac Typis Josephi Streater. Prostat apud plures Bibliopolas. Anno MDCLXXXVII.

```
//main program

 void draw () {
     background(30);
     for (int i = 0; i<N; i++) {

          Balls[i].newVelocity();
          Balls[i].newPosition();

     }
}
```

Recap

- Background
- Encapsulation
- Planet earth example
- Private and public
- EPD example
- Super- and subclass
- Cars example
- Bouncing balls revisited

thank you for your attention

TO DO (HOME EXERCISE)

```
//Your turn
```

Extend the car example and make it more exciting
(perhaps include collision detection)

```
thank you for your attention
```

Common Design Flaws

- Direct modification: classes that make direct modification of data values in other classes are a direct violation of *encapsulation*
- Too much responsibility: such classes are difficult to understand and use. Responsibility should be split into smaller meaningful packages
- No responsibility: such classes serve no purpose. Often arise when designers equate physical existence with logical design existence. “Money is no object”
- Classes with unused responsibility: Usually the results of designing software components without thinking about how they will be used
- Misleading names: Names should be short and unambiguously indicate what the responsibilities of the class involve
- Inappropriate inheritance: Occurs when subclassing is used in situations where the concepts do not share an “*is-a*” relationship.