EECS 1710
Programming for Digital Media

Lecture 23 :: Defining a Reference Type (Simple Classes)

YORK UNIVERSITÉ UNIVERSITY

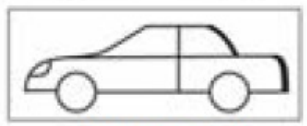
Recall: A class is an abstract view of a type

- An object has attributes[1], methods, an identity, and a state
- A class has attributes[1] and methods
- Objects with the same attributes and methods can be represented by a class that abstracts them:

Instantiate

Car:

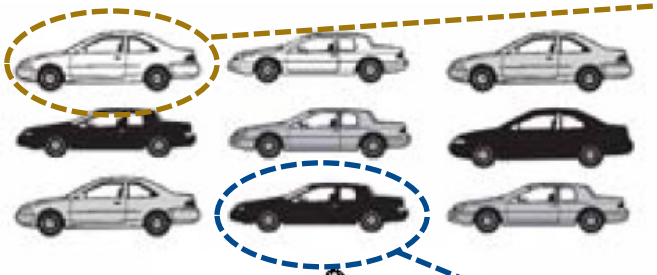
make:
model:
colour:
vinNumber:
year:



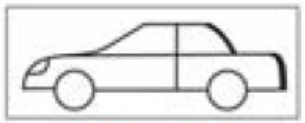
Abstract

An object is a concrete view of a type

- An object has attributes[1], methods, an identity, and a state
- A class has attributes[1] and methods
- Objects with the same attributes and methods can be represented by a class that abstracts them:



Instantiate



Abstract

Car:

make:	
model:	
colour:	
vinNumber:	
year:	

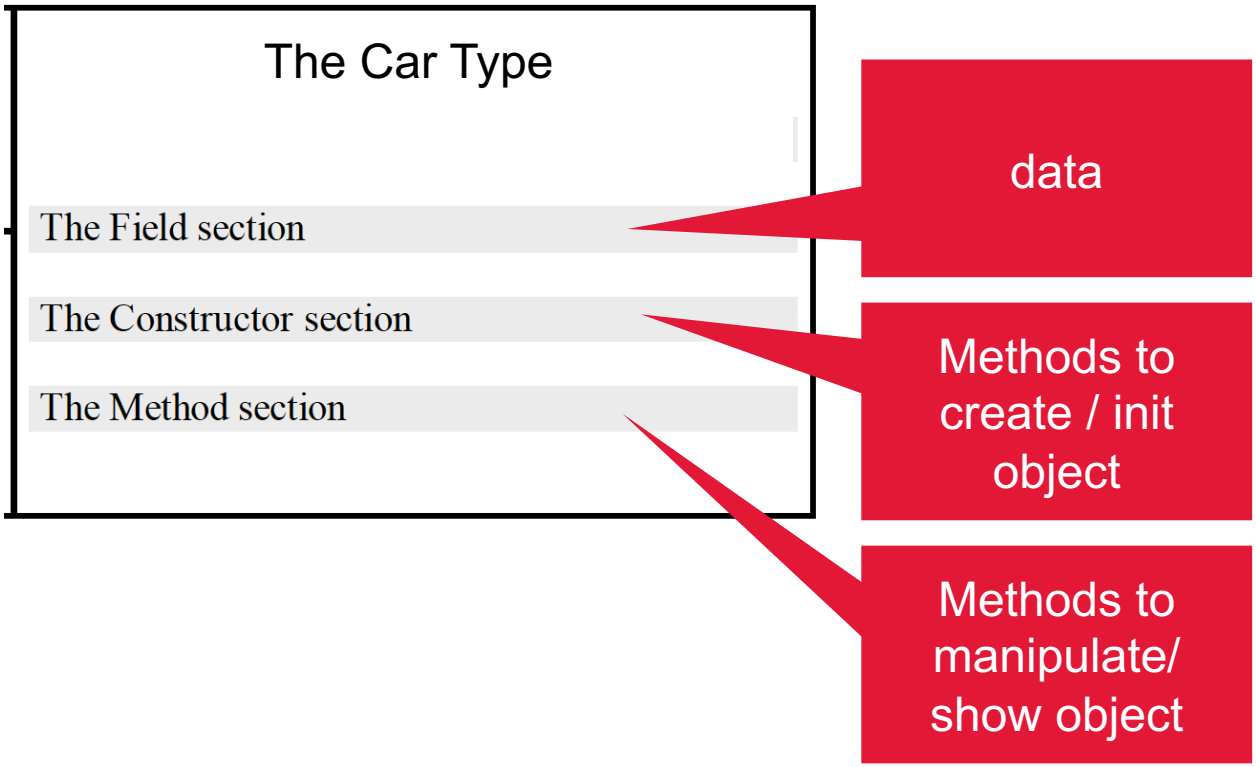
Car:

make:	Mazda
model:	CX5
colour:	blue
vinNumber:	R121FI23100
year:	2015

Car:

make:	Toyota
model:	Corolla
colour:	grey
vinNumber:	Z023BI44913
year:	2018

Objects typically define:
fields + constructors + methods



So, how do we make our own types ??

- We create an aggregate type with the keyword “class”

```
class MyType {

  // data fields



  // constructor(s)



  // method(s)

}
```

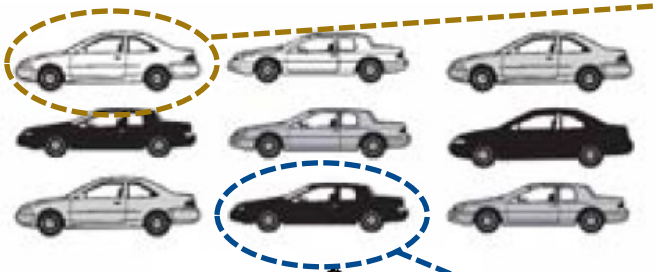
Two types of classes

- *Utility (static) – will look at these more in 1720*
  - *Includes fields/constants (usually constants)*
  - *Includes methods*
  - *Generally no constructors (cannot instantiate objects)*

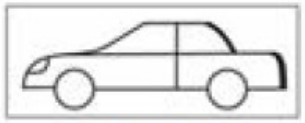
- **Dynamic (non-static)**
  - Includes fields/constants
  - Includes constructors (for initializing instantiated objects)
  - Includes methods

Example 1: Car Class

- An object has attributes[1], methods, an identity, and a state
- A class has attributes[1] and methods
- Objects with the same attributes and methods can be represented by a class that abstracts them:



Instantiate



Abstract

Car:

make:	
model:	
colour:	
vinNumber:	
year:	

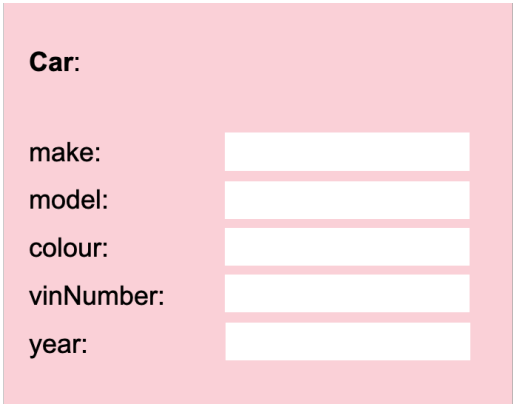
Car:

make:	Mazda
model:	CX5
colour:	blue
vinNumber:	R121FI23100
year:	2015

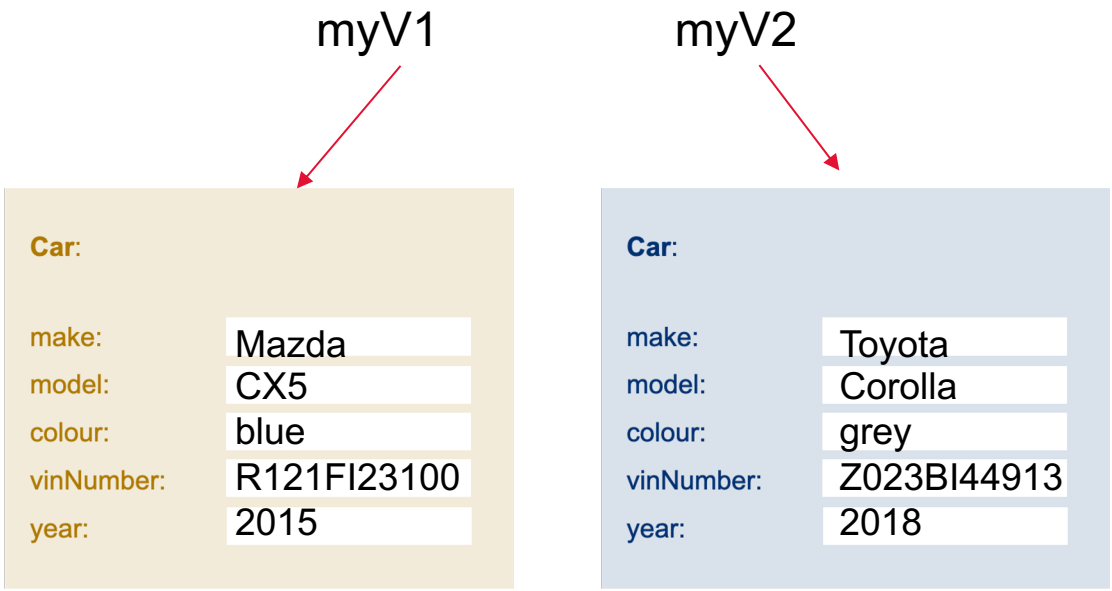
Car:

make:	Toyota
model:	Corolla
colour:	grey
vinNumber:	Z023BI44913
year:	2018

```
class Car {

  // data fields
  String make;
  String model;
  int colour;
  String vinNumber;
  int year;


  // constructor(s)
  Car(String mk, String mo, int col, String vin, int yr) {
    make = mk;
    model = mo;
    colour = col;
    vinNumber = vin;
    year = yr;
  }

  // method(s)
  void display() {
    println("make = " + make + ", model = " + model + ", year = " + year);
    println("colour = " + colour + ", vin = " + vinNumber);
  }


}
```



```
// CarExample

Car myV1;
Car myV2;


void setupCar() {
  // called from setup()

  myV1 = new Car("Mazda", "CX5", color(0,255,0), "R121FI23100", 2015);
  myV2 = new Car("Toyota", "Corolla", color(100,100,100), "Z023BI44913", 2018);

}


void drawCar() {
  // assume called from draw()

  myV1.display();
  println();
  myV2.display();


}
```



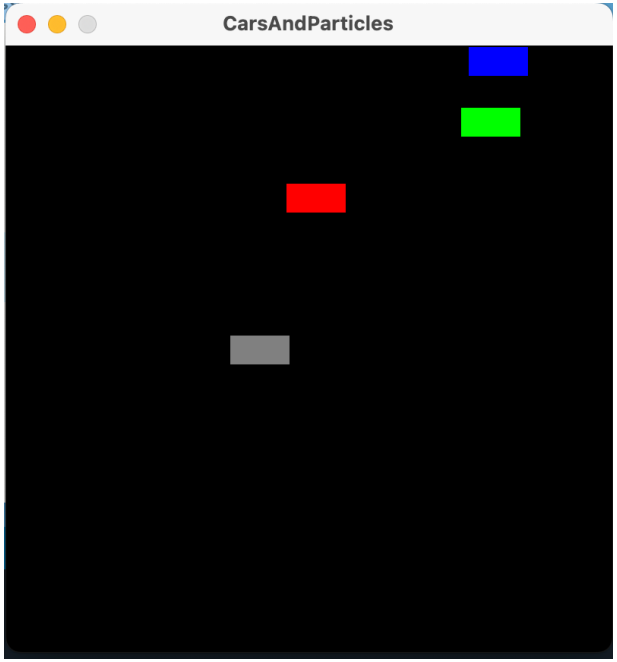
```
// CarExample

Car myV1;
Car myV2;


void setupCar() {
  // called from setup()

  myV1 = new Car("Mazda", "CX5", color(0,255,0), "R121FI23100", 2015);
  myV2 = new Car("Toyota", "Corolla", color(100,100,100), "Z023BI44913", 2018);

}


void drawCar() {
  // assume called from draw()

  myV1.display();
  println();
  myV2.display();


}
```

Console output:

```
make = Mazda, model = CX5, year = 2015
colour = -16711936, vin = R121FI23100

make = Toyota, model = Corolla, year = 2018
colour = -10197916, vin = Z023BI44913
```

Example 2: MovingCar (animated object)

- Think 2D “crossy road” or “frogger”

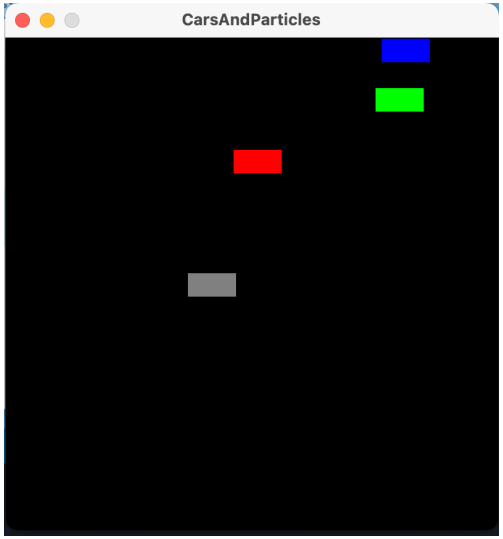


- Car constructor(s)
  - So we can instantiate several car objects
  - How to make default vs. custom car?
- Car needs to have a colour, lets represent as a filled rect
  - Needs to have a field to store color
- Car needs to move across screen (e.g. left to right)
  - Needs to have fields for position (possibly size)
  - Needs to have fields for velocity/speed (if we want different instances of cars to move with different speeds)
  - Needs a method to make it move

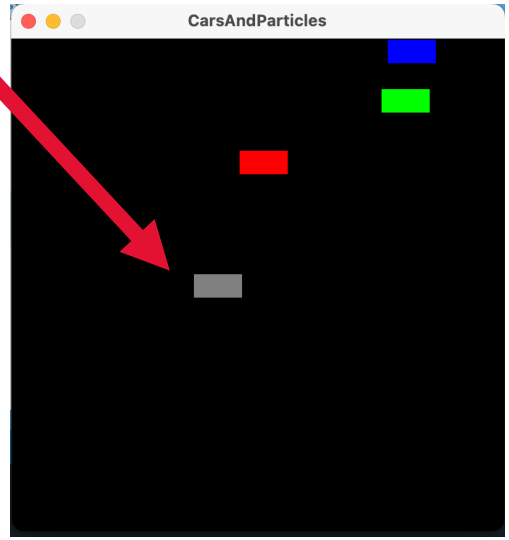
Constructors

- Default / No-Argument Constructor
  - Takes no arguments
  - Use to set default values
  - Should always include… if you don't, one is usually assumed
    - Be careful, if any fields are reference types, it wont initialize them (however if primitive types, they usually get initialized… e.g. numerical fields usually initialized with 0)
- Custom Constructor(s)
  - Takes parameters/arguments
  - Can create multiple versions (as long as they have different signatures)
  - They should use the arguments to set/initialize fields

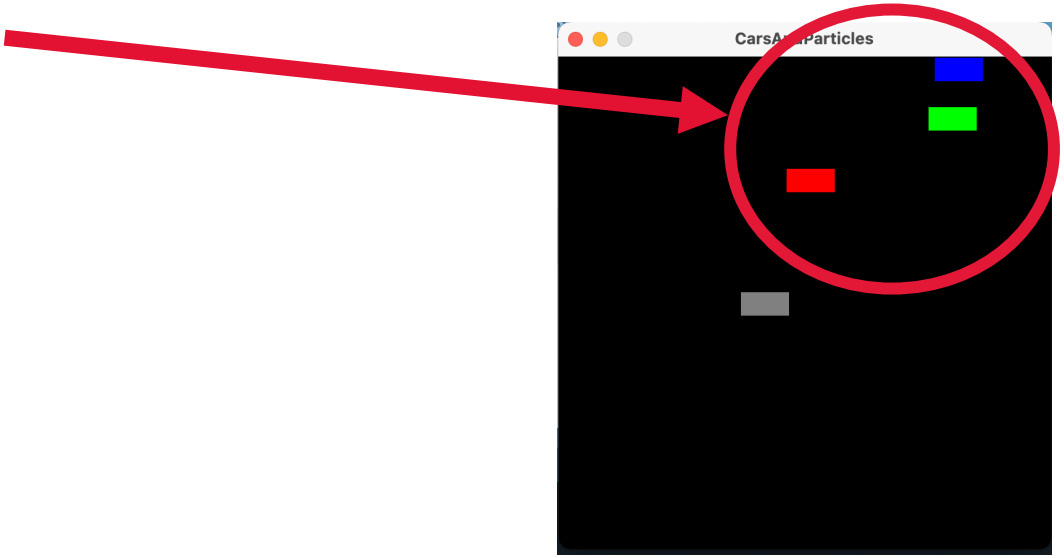
```
class MovingCar {

  // data fields
  color c;
  float xpos;
  float ypos;
  float xspeed;









}
```



```
class MovingCar {

  // data fields
  color c;
  float xpos;
  float ypos;
  float xspeed;

  // a default constructor (no parameters, set fields with default values)
  MovingCar() {
    c = color(128,128,128);    // grey car
    xpos = 0;
    ypos = height/2;
    xspeed = 1;
  }

}
```



```
class MovingCar {

  // data fields
  color c;
  float xpos;
  float ypos;
  float xspeed;

  // a default constructor (no parameters, set fields with default values)
  MovingCar() {
    c = color(128,128,128);    // grey car
    xpos = 0;
    ypos = height/2;
    xspeed = 1;
  }

  // a custom constructor
  MovingCar(color tempC, float tempXpos, float tempYpos, float tempXspeed) {
    c = tempC;
    xpos = tempXpos;
    ypos = tempYpos;
    xspeed = tempXspeed;
  }




}
```



Constructor **signatures** :

```
MovingCar()

MovingCar(color tempC, float tempXpos, float
tempYpos, float tempXspeed)
```

A note on constructor **signatures** :

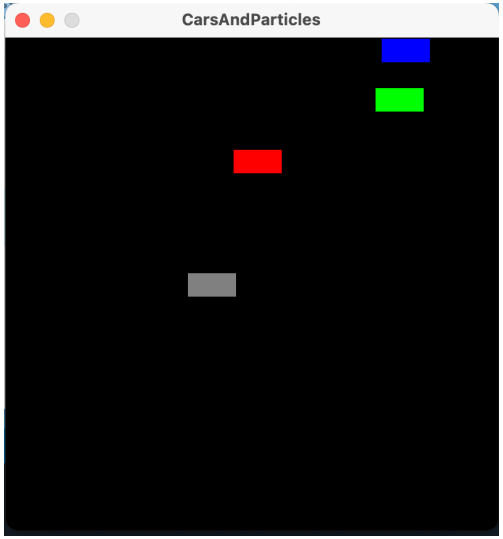
```
MovingCar()

MovingCar(color tempC, float tempXpos, float tempYpos, float
tempXspeed)

// add more custom constructors? Sure… as long as no
// two constructors have the same signature
MovingCar(color tempC)
MovingCar(color tempC, float tempXspeed)

// e.g. what if you want another to init tempXpos?
```

A note on constructor **signatures** :

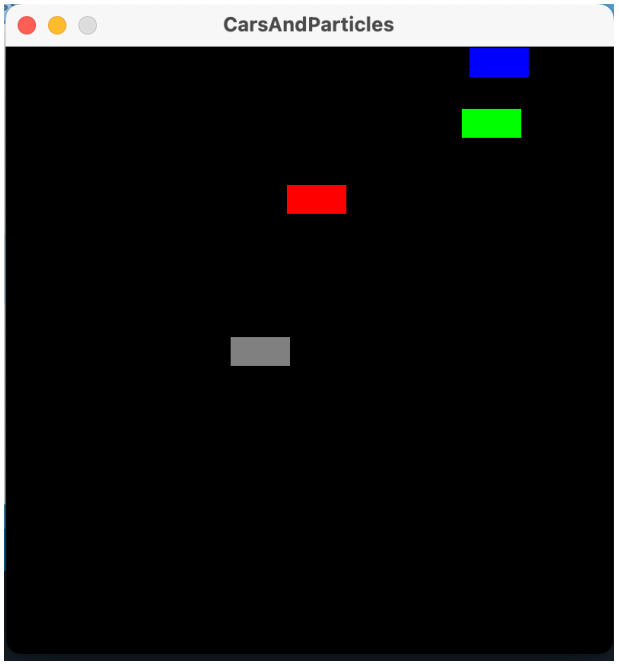
```
MovingCar()

MovingCar(color tempC, float tempXpos, float tempYpos, float
tempXspeed)

// add more custom constructors? Sure… as long as no
// two constructors have the same signature
MovingCar(color tempC)
MovingCar(color tempC, float tempXspeed)

// e.g. not be allowed if you already have the above:
MovingCar(color tempC, float tempXpos)
```

Same signatures:
MovingCar (color, float)

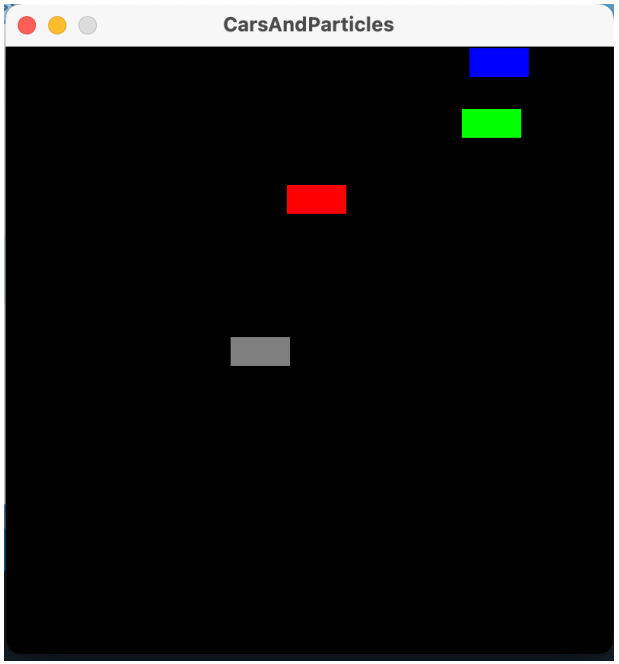
```
class MovingCar {

  // data fields
  color c;
  float xpos;
  float ypos;
  float xspeed;

  // a default constructor (no parameters, set fields with default values)
  MovingCar() {
    c = color(128,128,128);    // grey car
    xpos = 0;
    ypos = height/2;
    xspeed = 1;
  }

  // a custom constructor
  MovingCar(color tempC, float tempXpos, float tempYpos, float tempXspeed) {
    c = tempC;
    xpos = tempXpos;
    ypos = tempYpos;
    xspeed = tempXspeed;
  }
  // method to draw car
  void display() {
    stroke(0);
    fill(c);
    rectMode(CENTER);
    rect(xpos,ypos,40,20);
  }
}
```



```
class MovingCar {

  // data fields
  color c;
  float xpos;
  float ypos;
  float xspeed;

  // constructors (implementation not shown)
  MovingCar() {…}
  MovingCar(color tempC, float tempXpos, float tempYpos, float tempXspeed) {…}

  void display() {
    stroke(0);
    fill(c);
    rectMode(CENTER);
    rect(xpos,ypos,40,20);
  }

  void move() {
    xpos = xpos + xspeed;
    if (xpos > width) {
      xpos = 0;
    }
  }

}
```



```
class MovingCar {

  // data fields
  color c;
  float xpos;
  float ypos;
  float xspeed;

  // constructors (implementation not shown)
  MovingCar() {…}
  MovingCar(color tempC, float tempXpos, float tempYpos, float tempXspeed) {…}

  void display() {
    stroke(0);
    fill(c);
    rectMode(CENTER);
    rect(xpos,ypos,40,20);
  }

  void move() {
    xpos = xpos + xspeed;
    if (xpos > width) {
      xpos = 0;
    }
    if (xpos < 0) {
      xpos = width;
    }
  }

}
```

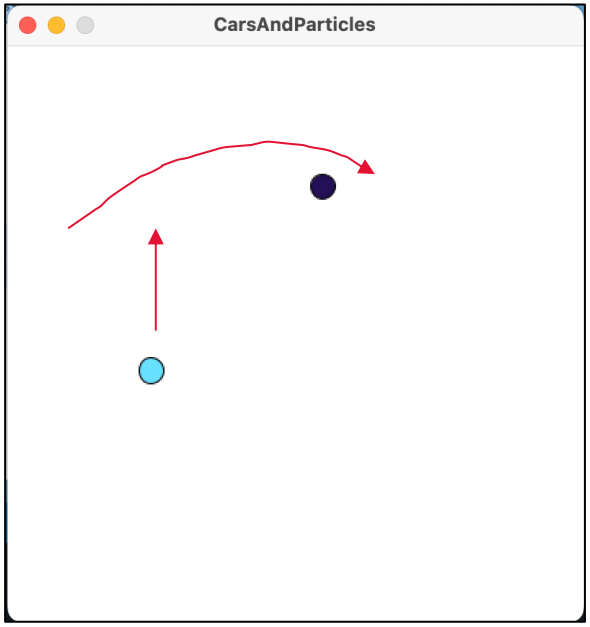


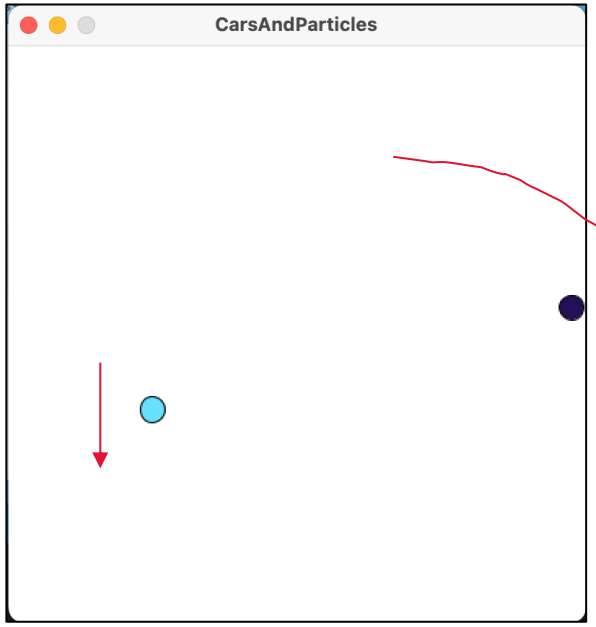
MovingCar Example:

```
MovingCar mvCar0;
MovingCar mvCar1;
MovingCar mvCar2;
MovingCar mvCar3;

// called from setup()
void setupMovingCar() {
  mvCar0 = new MovingCar();
  mvCar1 = new MovingCar(color(255,0,0),0,100,4);
  mvCar2 = new MovingCar(color(0,0,255),0,10,2);
  mvCar3 = new MovingCar(color(0,255,0),0,50,-3);

}

// called from draw()
void drawMovingCar()  {
  background(0,0,0);
  mvCar0.move(); mvCar0.display();
  mvCar1.move(); mvCar1.display();
  mvCar2.move(); mvCar2.display();
  mvCar3.move(); mvCar3.display();
}
```

Particle(s)

- Inspired by our projectile motion example from earlier lectures…





```
final float GRAVITY = 9.8;
final float DT = 0.1;

class Particle {

  PVector pos;
  PVector vel;
  color col;
  float radius;

  Particle(float x, float y, float dx, float dy, color c, float r) {
    pos = new PVector(x,y);
    vel = new PVector(dx,dy);
    col = c;
    radius = r;
  }

  void display() {
    fill(col);
    ellipseMode(RADIUS);
    circle(pos.x, pos.y, radius);
    stroke(0,0,0);
  }

  void move() {
    pos.x = pos.x + vel.x*DT;
    pos.y = pos.y + vel.y*DT;
    vel.y = vel.y + 0.5*GRAVITY*DT*DT;  // includes acceleration term
  }
}
```

```
Particle bullet;
Particle firework;

void setupParticle() {

  bullet = new Particle(0,height/2,10,-10,
                        color(random(255),random(255),random(255)),
                        random(20));

  firework = new Particle(width/4,  height,
                     0,-10.0+random(10)-10.0,
                     color(random(255),random(255),random(255)),
                     random(20) );


}



void drawParticle() {
  background(255,255,255);
  bullet.display();
  bullet.move();

  firework.display();
  firework.move();

}
```

Next (final) lecture

- Eclipse toolset
  - Installing Eclipse
  - Add Proclipsing to Eclipse (Eclipse plugin for Processing)
  - Basic Processing project

- Java Anatomy Preview (non-processing version of things)