Processing Paris #02

22/23 Avril 2011

La Fonderie de l’image

Variable

- Type
  - `int`, `float`, `char`, `String`, `PVector`, `PImage`, ...
- Nom de la variable
- Affectation (symbole “=”)

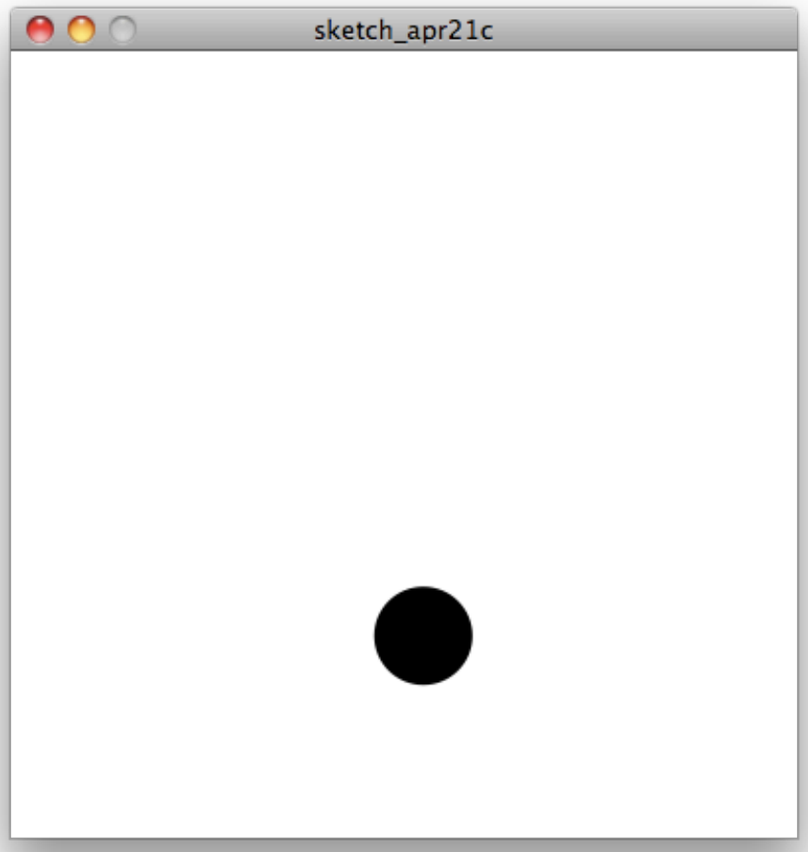
Variable

- Exemples
  - `float x = 3.14;`
  - `int index = 24;`
  - `String maChaine = “Processing!”;`
  - `char c = ‘A’;`

Classe

- Canevas d’un objet
- Description
  - propriétés (“membres” d’une classe)
  - comportement (“méthodes” d’une classe)
- Définition d’un nouveau type de données utilisable dans notre sketch

Classe



- Comment décrire cet objet ?

Classe

- class Particle{
  ...
  }; // point virgule optionnel
- “Particle” est le nom de la classe

Classe

- Caratéristique d’une particule
  - position sur l’écran
    - traduire en propriétés x et y
- action de dessin
  - traduire en une méthode draw()
- d’autres caractéristiques ?

Classe

définition

- ```
  class Particle{
      float x; // propriété
      float y; // propriété
      void draw(){ // méthode
        ellipse(x,y,50,50);
      }
  };
  ```
  ```

Classe

Constructeur

- Membre spécial de la classe permettant d'initialiser les propriétés de l'objet (= personnalisation de cet objet)
- Possibilité d'avoir plusieurs constructeurs par classe.
- ```
  class Particle{
      Particle(float x, float y){
        this.x = x; // sauvegarde dans la propriété de la valeur passée en param.
        this.y = y; // idem
      }

      float x; // propriété
      float y; // propriété

      void draw(){ // méthode
        ellipse(x,y,50,50);
      }
  };
  ```
  ```

Classe

Instanciation

- Instanciation = action de créer un objet à partir de sa classe (canevas)
- Mot clé `new` qui déclenche l’appel au constructeur de la classe.

Classe

Instanciation

- `Particle p = new Particle(width/2, height/2);`

Classe

Instanciation multiple (méthode “naïve”)

- Dans le setup()

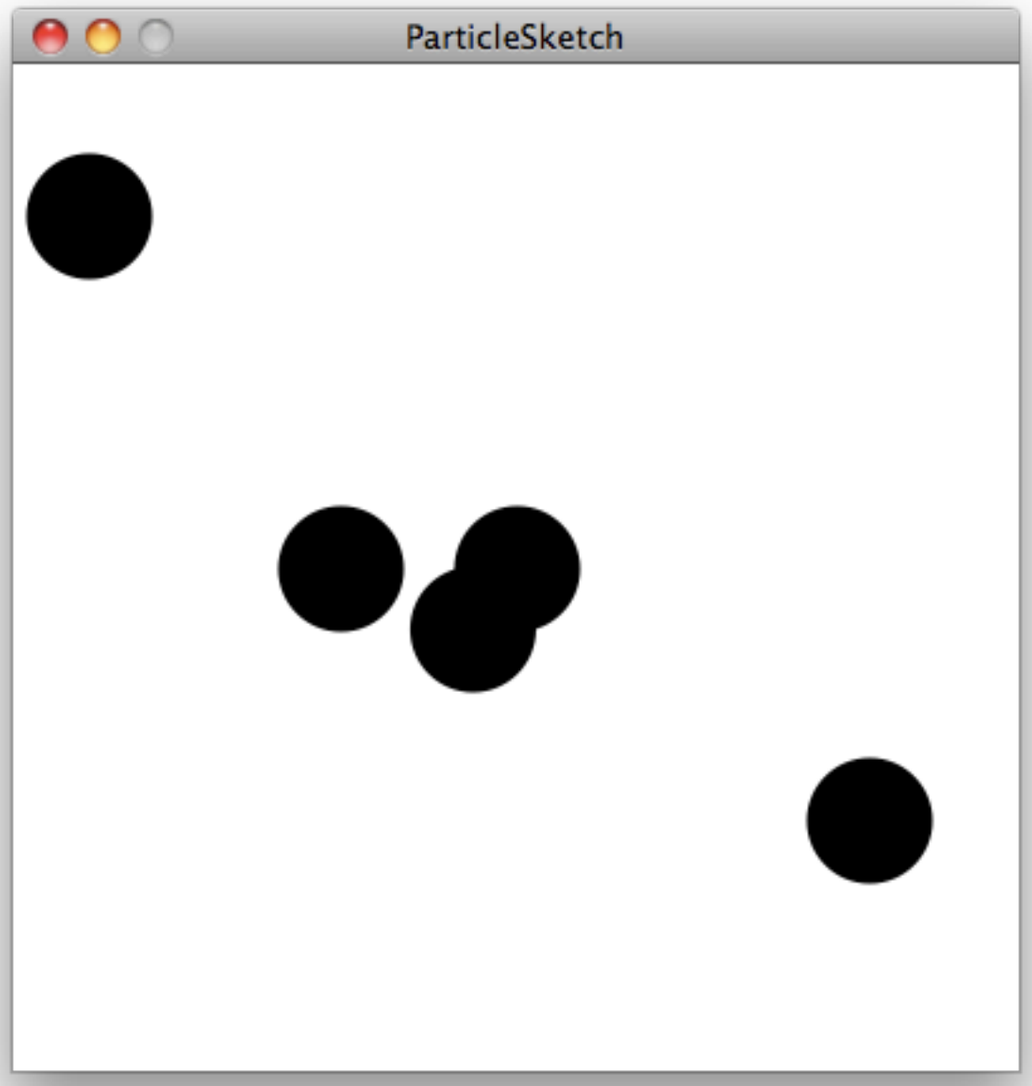
```
Particle p  = new Particle(width/2, height/2);
Particle p1 = new Particle(30, 60);
Particle p2 = new Particle(130, 200);
Particle p3 = new Particle(random(width), random(height));
Particle p4 = new Particle(340, 300);
```

- Puis dans la fonction draw()

```
p.draw();
p1.draw();
p2.draw();
p3.draw();
p4.draw();
```

Classe

Instanciation multiple (méthode “naïve”)



Classe

Instanciations multiples

- Quid si nous voulons générer 100 particules ? Impensable de répéter 100 fois la même ligne.
- Les tableaux (statiques et dynamiques) viennent à la rescousse.

Tableaux

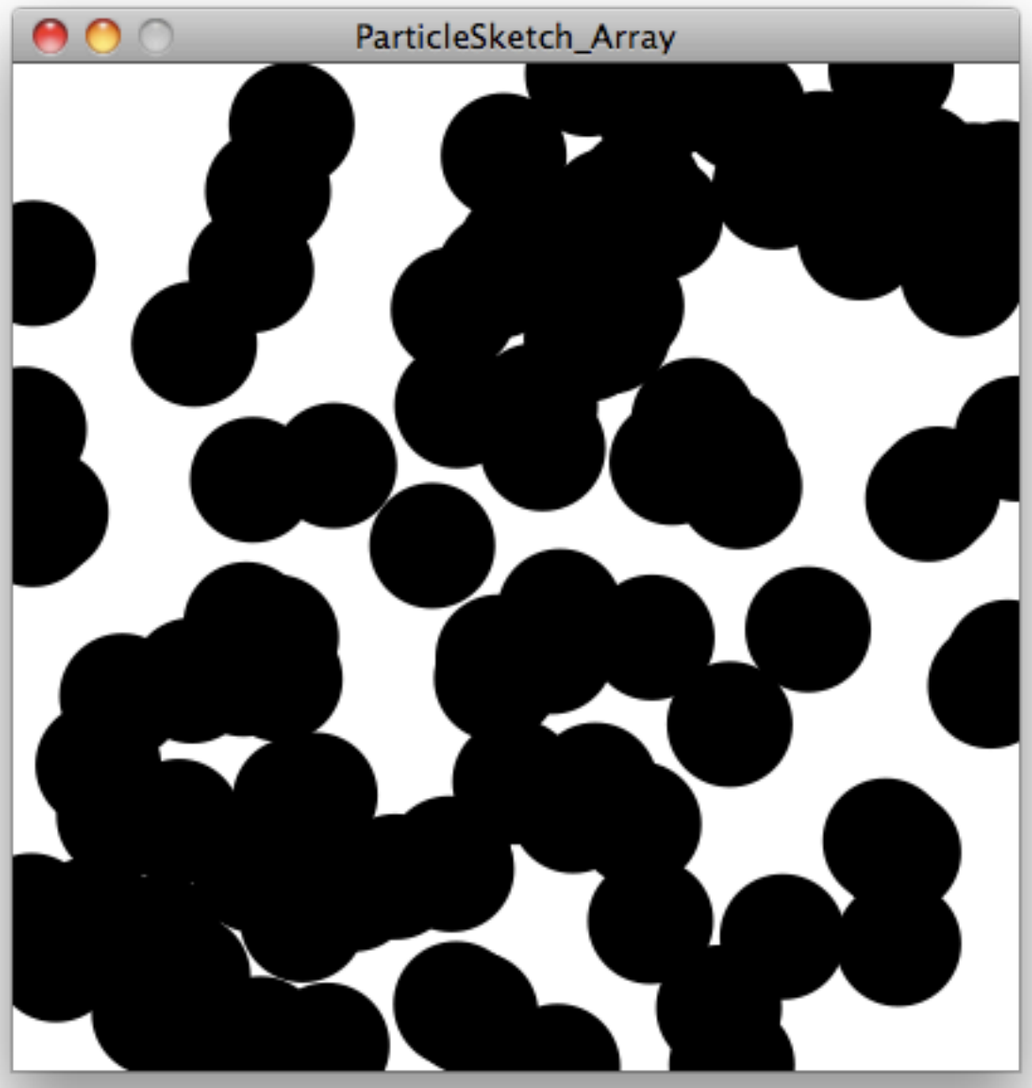
- Déclaration
  - Particle[] particles;
- Création du tableau (mot clé new)
  - particles = new Particle[100]; // entre crochets, le nombre de particules
- Remplissage à l’aide d’une boucle for
  - ```
    for (int i=0;i<100;i++){
        particles[i] = new Particle(random(width), random(height));
    }
    ```
    ```

Tableaux

Parcours

- Ne pas oublier la boucle pour le dessin dans la fonction draw() du sketch !
- ```
  for (int i=0;i<100;i++){
      particles[i].draw(); // appel de la méthode draw(), propre à chaque objet
  }
  ```
  ```

Tableaux



Tableaux & Objets

Un peu d’animation

- Comment rajouter une propriété à tous nos objets d’un seul coup ?
- Facile, il suffit de simplement modifier la définition de la classe des objets (appelés aussi instances)
- Par exemple, nous allons rajouter des propriétés de vitesse et une action de mouvement, qui sera effectuée avant le dessin.

Tableaux & Objets

Modification de la classe

- Modification de la classe => report sur tous les objets.

- ```
  class Particle{
      Particle(float x, float y){
        this.x = x; // sauvegarde dans la propriété de la valeur passée en param.
        this.y = y; // idem
        this.vx = random(-10,10);
        this.vy = random(-10,10);
      }

      float x; // propriété
      float y; // propriété
      float vx, vy; // ajout de la vitesse

      void move(){ // méthode
        x += vx;
        y += vy;
      }

      void draw(){ // méthode
        ellipse(x,y,50,50);
      }
  };
  ```
  ```

Tableaux & Objets

Modification de la classe : mieux!

- “compactage” des données en utilisant la classe PVector.
-
```
class Particle{
    Particle(float x, float y){
      pos = new PVector(x,y);
      speed = new PVector( random(-10,10), random(-10,10) );
    }

    PVector pos;
    PVector speed;

    void move(){ // méthode
      pos.add(speed);
    }

    void draw(){ // méthode
      ellipse(pos.x,pos.y,50,50);
    }
};
```

Tableaux dynamiques

- permet de stocker des objets de même type dans une liste. On est plus obligé de prévoir le nombre d’éléments à l’avance.
- facilité pour rajouter ou enlever à la volée des éléments.
- Type `ArrayList`

Tableaux dynamiques

Déclaration

- Type, nom, affectation
- `ArrayList myList = new ArrayList();`

Tableaux dynamiques

Ajout

- méthode “add(Object obj)”
  - rajoute un objet à la fin de la liste.
- myList.add(new Particle(60,80));

Tableaux dynamiques

Retrait

- méthode “remove(int index)”
- retourne l’objet qui a été enlevé de la liste.
- Attention ! “index” doit être valide sinon Exception.

Tableaux dynamiques

Retrait

- `Particle p = (Particle)myList.remove(0);`
- à exécuter si la liste myList contient au moins un élément ...!

Tableaux dynamiques

Parcours

- 1ère méthode : boucle for et index

```
Particle p;
for (int i=0;i<myList.size();i++){
    p = (Particle) myList.get(i);
    ...
}
```

Tableaux dynamiques

Parcours

- 2ème méthode : itérateur

```
Iterator it = myList.iterator();
for ( ;it.hasNext(); ){
    p = (Particle) it.next();
    ...
}
```

Tableaux dynamiques

les ‘generics’

- syntaxe “pratique”
- Disponible à partir de Java 1.5

Tableaux dynamiques

les ‘generics’

- `ArrayList<Particle> myList = new ArrayList<Particle>();`

Tableaux dynamiques

les ‘generics’

- Déclaration
  - `ArrayList<Particle> myList = new ArrayList<Particle>();`
- Ajout
  - `myList.add ( new Particle(random(width), random(height) );`
- Retrait
  - `Particle p = myList.remove(0); // plus besoin de cast !`

Tableaux dynamiques

les ‘generics’ : parcours

- ```
  for (Particles p:myList){
  ...
  }
  ```
  ```