IAT 265 Lecture 7
Key Interactions and ArrayList

Topics

- Recap: Polymorphism
- Key Interactions
  - keyPressed, key & keyCode variables
  - Key event handlers
  - Key mapping for multi-key Interactions
- Java collection class: ArrayList

Recap: Polymorphism

- **Polymorphism**
  - the ability to create a variable, a method, or an object that has more than one form
  - Two types:
    - **Overriding polymorphism**
    - **Inclusion polymorphism**

Polymorphism works in context of Inheritance

- Subclasses inherit fields and methods from parent
- Subclasses extend or overwrite capabilities of parent

```
class EatingBug extends Bug{
  …
}
```

Overriding polymorphism

- A subclass replaces the implementation of one or more of its parent's methods (with same signatures)

- Can EatingBug has its own drawBug() method, so that it draws an EatingBug object differently?
  - Yes, and when drawBug() gets called at runtime, it overrides its parent version

Override Bug's drawBug() to draw it in a different color

```
//Override parent's drawBug method
  void drawBug() {
    //call parent's drawBug() method to draw
    // a regular bug
    super.drawBug();

    //Draw on top of parent's version
    pushMatrix();
    translate(bugX, bugY);
    if(alive) {
      //make the bug rotate
      if( changeX <0 ) {
        rotateY(PI);
      }
      //redraw body with orange color
      fill(165, 0, 0);
      ellipseMode(CORNER);
      ellipse(0, 0, bSize, bSize);  ////draw ladybug

      //redraw the dots and head in black color
      fill(0);
      ellipse(bSize/5, bSize/7, bSize/6, bSize/6);
      ellipse(bSize/5, bSize*5/7, bSize/6, bSize/6);
      ellipse(bSize*3/5, bSize/7, bSize/6, bSize/6);
      ellipse(bSize*3/5, bSize*5/7, bSize/6, bSize/6);

      ellipseMode(CENTER);
      arc(bSize, bSize/2, bSize/4, bSize/4, -PI/2, PI/2);

      //redraw the body line with black color
      stroke(0);
      line (0, bSize/2, bSize, bSize/2);
    }
    popMatrix();
  }
```

Inclusion polymorphism

- A variable or parameter of superclass can denote objects of its subclasses

- So it is perfectly legal to do this:

```
Bug bug = new EatingBug(random(gardenW),
  random(gardenH), random(-1,1), random(-1,1),
  random(12,36));
```

- Or mix objects of different types in one array:

```
Bug[] bugs = new Bugs[10];
bugs[0] = new Bug (random(width), random(height),
  random(-1,1), random(-1,1), random(12,36));
bugs[1] = new EatingBug (width/2, height/2, 1, 1, 20);
```

Then how to differentiate objects of different types in a mixed array?

- Use instanceof operator & type casting

```
if(bugs[i] instanceof EatingBug) {
      EatingBug eatBug = (EatingBug) bugs[i];
      //from this point on you can call EatingBug's methods
      if(eatBug.detectCollision(bugs[k]) &&
           eatBug.checkHeadOn(bugs[k])) {
           eatBug.eat(bugs[k]);
       ...
}
```

Same goes for parameters ...

- A parameter of a superclass type can accept its subclass objects as arguments
  - This is useful when you have more than one subclass

```
void eat(Bug otherBug) {
  if(bSize > otherBug.bSize) {
    //grow itself by 10%
    bSize *= 1.1;
    //kill the other bug
    otherBug.alive = false;
  }
```

Here you can pass in object of any Bug's subclasses as its argument, e.g. object of EatingBug, VeggieBug (if we define such a subclass of Bug)

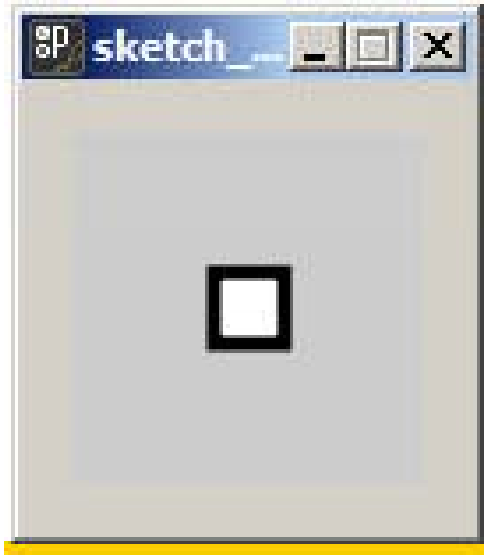
Keyboard Interactions

- Processing registers the most recently pressed key and whether a key is currently pressed
  - The boolean variable ***keyPressed*** is *true* if a key is pressed and *false* if not
  - ***keyPressed*** remains *true* while the key is held down and becomes *false* only when the key is released

```
//draw a rectangle while any key is pressed

void setup()
{
  size(100,100);
  smooth();
  strokeWeight(4);
}
void draw()
{
  background(204);
 if (keyPressed==true)
 {

   rect(40,40,20,20);
 }
   else
   {
     line(20,20,80,80);
   }
 }
```



key variable – which key is pressed?

- The key variable (of char type) stores the most recently pressed or released key
  - Commonly used for keys included in the ASCII specification (e.g. a~z, A~Z, ENTER/RETURN, ESC,...)

```
void draw() {
    if(keyPressed) {
        if (key == 'b' || key == 'B' ) {
               fill(0);
        }
     } else {
        fill(255);
     }
     rect(25, 25, 50, 50);
}
```

keyCode variable – which coded key is pressed?

- The keyCode variable is used to detect special keys such as the UP, DOWN, LEFT, RIGHT arrow keys and ALT, CONTROL, SHIFT

  - When checking for these keys, it's necessary to check first if the key is coded, with the conditional "if (key == CODED)"

```
if (key == CODED) {
        if (keyCode == UP) {
                fillVal = 255;
        } else if (keyCode == DOWN) {
                fillVal = 0;
}
```

Key events callbacks

- ***keyPressed():*** called once every time a key is pressed

- ***keyReleased():*** called once every time a key is released

Key Mapping for Multi-key Interactions

```
// Key Mapping for Multi-key interactions
boolean downKey, upKey, leftKey, rightKey;
void keyPressed() {
  if (key == CODED && keyCode == RIGHT)   rightKey = true;
  if (key == CODED && keyCode == LEFT)   leftKey = true;
  if (key == CODED && keyCode == UP)   upKey = true;
  if (key == CODED && keyCode ==DOWN)   downKey = true;
}
void keyReleased() {
  if (key == CODED && keyCode == RIGHT)   rightKey = false;
  if (key == CODED && keyCode == LEFT)   leftKey = false;
  if (key == CODED && keyCode == UP)   upKey = false;
  if (key == CODED && keyCode ==DOWN)   downKey = false;
}
```

Case study: Key-controlled Avatar

```
class AvatarBug extends Bug {
  AvatarBug(float x, float y, float chgX, float chgY, float sz) {
    super(x, y, chgX, chgY, sz);
  }
  //method eat: eat the other bug if bigger otherwise kill itself
  void eat(Bug otherBug) {
    if(bSize > otherBug.bSize) {
     bSize *= 1.1;  //grow itself by 10%
     otherBug.alive = false;   //kill the other bug
    }   else  {
      this.alive = false;        //otherwise kill itself
      drawWaves();               //draw waves to show being killed
    }
  }
```

Case study: Key-controlled Avatar (2)

```
//method for drawing waves
 void drawWaves() {
  stroke(200, 0, 0);
  noFill();
  for(int i=1; i<=2; i++) {
    ellipse(bugX, bugY, i*bSize, i*bSize);
  }
 }

 //methods checkHeadOn() & drawBug()
 //are the same as EatingBug
 ...
```

```
//methods for move left, rigth, up & down
void moveRigtht(){
  if(changeX < 0)   changeX *= -1;
  bugX += changeX;
}

void moveLeft(){
  if(changeX > 0)    changeX *= -1;
  bugX += changeX;
}

void moveUp(){
  bugY -= changeY;
}

void moveDown(){
  bugY += changeY;
 }
}
```

Case study: Instantiation in the setup & draw functions

```
Bug[] bugs = new Bug[count];
AvatarBug avtBug;

  //Key Mapping for Multi-key Interactions
  //(Exactly the same as on page 14)
  boolean downKey, upKey, leftKey, rightKey;
  void keyPressed() { ...
  }
  void keyReleased() { ...
  }

//method respawn itself at the center
AvatarBug respawn() {
  return new AvatarBug(width/2, height/2, 4,
     4, 20);
}
```

```
void setup() {
  ...
  avtBug = respawn();
}
void draw() {
  ...
  //move avatar based on keypressed
  if (rightKey) avtBug.moveRigtht();
  if (leftKey) avtBug.moveLeft();
  if (upKey) avtBug.moveUp();
  if (downKey) avtBug.moveDown();

   //nested for loops i & k
   ...
   if(bugs[k].alive &&
       avtBug.detectCollision(bugs[k])  &&
       avtBug.checkHeadOn(bugs[k])) {
        avtBug.eat(bugs[k]);
        if(!avtBug.alive) {
          avtBug = respawn();
        }
   } }
```

How do we make objects disappear when destroyed?

- So far we have used conditional drawing
  - E.g. based on Bug's *alive* status, draw the bug only when it is true

```
void drawBug() {
   pushMatrix();
   translate(bugX, bugY);
   if(alive) {    //draw only if the bug is alive
       fill(0, 0, 60);
       ellipse(0, 0, bSize, bSize);
            ...
    }
    popMatrix();
}
```

- This is actually not the best approach, as the destroyed objects, although invisible, still sit in the memory

A better way: use ArrayList

- Java comes with thousands of classes in the Java Platform API
- Documentation is available on Sun's website
  - http://download.oracle.com/javase/6/docs/api/
- Let's look at ArrayList – a Java collection class

ArrayList

- It's a resizable list
  - Can add and delete things without worrying about declaring the size
- The main methods we care about are add(), get(), and remove(), and size()
- Steps in using ArrayList
  - Declare a variable of type ArrayList
  - Create a new ArrayList and assign it to the variable
  - Call add(), get() and remove() and size() on ArrayList as you need them

Using `ArrayList.add()`

- The argument type of the add method is Object
  - Object is the parent class of *all classes* in Java
  - With a parameter of Object type, you can pass in an object of any class

- So, to initialize our asteroids...

```
ArrayList bugs = new ArrayList();
for(int i = 0; i < count; i++){
   bugs.add(new Bug(
    random(width),random(height), random(1,1),
    random(-1,1), random(12,36));
}
```

Getting things out of an ArrayList

- **`ArrayList.get(int i)`** – returns the *ith* object (starting with 0)

- But this doesn't work!

  `bugs.get(i).drawBug();`

  Why?

Need to cast back from Object

- Since things are put in an ArrayList as Object, they come back out as Object
  - It's like they forget their more detailed type
  - So, when using ArrayList (or any Java collection class), you need to cast back to the more detailed type
- For our Bug example:

```
Bug bugi = (Bug)bugs.get(i);
```

```
//For the rest of our previous case study,
//just replace all bugs[i] with bugi,
//and it will do the same job as before
```

Destroying bugs

- When a Bug is eaten by an AvatarBug, we need to destroy it
  - This was one of the major reasons for using ArrayList

```
void destroy(ArrayList bugs) {
  bugs.remove(this);
}
```

- By doing this, we don't need to check alive status for Bug objects, as any dead bug would be removed → doesn't exist anymore
  - AvatarBug still needs to check alive status to respawn, so make it a field of AvatarBug only

Call destroy() method in AvatarBug

```
//method eat: eat the other bug if bigger otherwise kill itself
  void eat(Bug otherBug) {
    if(bSize > otherBug.bSize) {
      //grow itself by 10%
      bSize *= 1.1;

      //kill the other bug
      //Don't need this anymore: otherBug.alive = false;
      otherBug.destroy(bugs);
    }
    else {
      this.alive = false;
      drawWaves();
    }
```

Summary

- Recap: Polymorphism
- Key Interactions
  - keyPressed, key & keyCode variables
  - Key event handlers
  - Key mapping for multi-key Interactions
- Java collection class: ArrayList