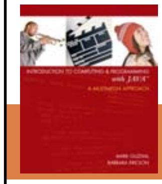
TOPIC 13

OBJECTS IN MEMORY, PASSING PARAMETERS



Notes adapted from Introduction to Computing and Programming with Java: A Multimedia Approach by M. Guzdial and B. Ericson, and instructor materials prepared by B. Ericson.

Outline

- How objects are stored in memory
- How parameters are passed to methods

Objects in Memory

- Recall the Student class with two constructors defined as follows:

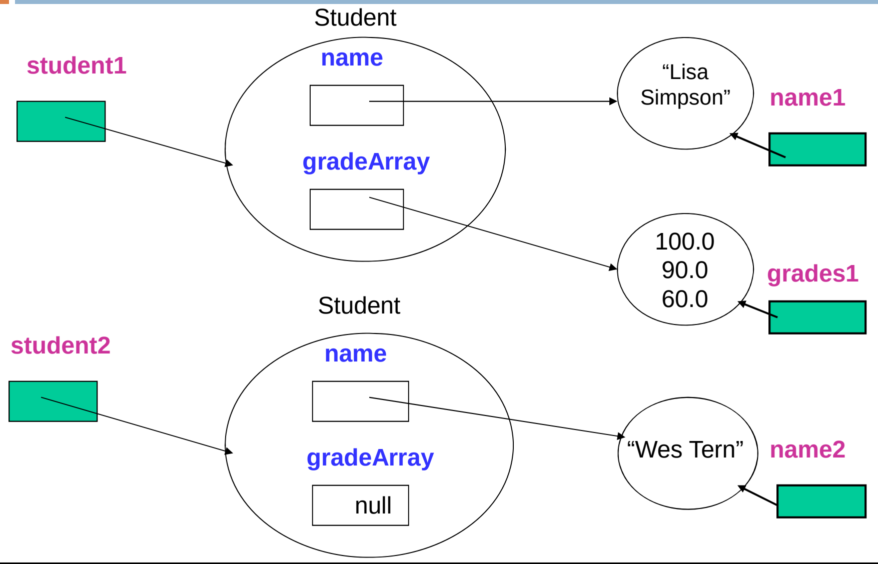
```
public Student(String theName)
{
  this.name = theName;
}
public Student(String theName, double[] grades)
{
   this.name = theName;
   this.gradeArray = grades;
}
```

Objects in Memory

- Here is what happens in memory when the following code segment is executed

```
String name1 = “Lisa Simpson”;
String name2 = “Wes Tern”;
double [] grades1 = {100.0, 90.0,60.0};
Student student1 = new Student(name1,grades1);
Student student2 = new Student(name2);
```

Objects in Memory: a Trace



Trace

- Recall that a reference variable "points to" an object, i.e. it contains the "location" of the object
- So, for student1
  - The name field contains the location of the string pointed to by name1
  - The gradeArray field contains the location of the array pointed to by grades1
- For student2, gradeArray is null

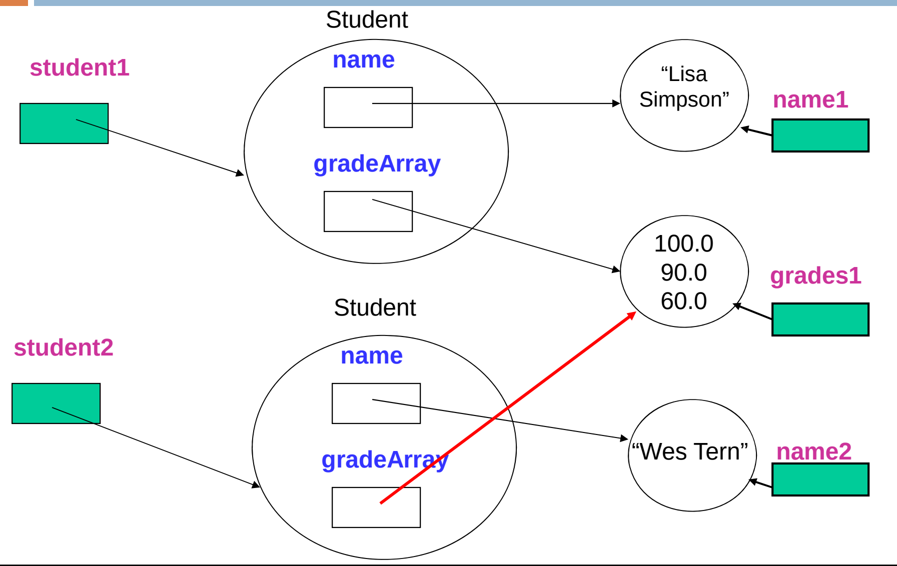
Objects in Memory

7

- Now consider the following code segment instead

```
String name1 = “Lisa Simpson”;
String name2 = “Wes Tern”;
double [] grades1 = {100.0, 90.0,60.0};
Student student1 = new Student(name1,grades1);
Student student2 = new Student(name2,grades1);
```

Objects in Memory: a Trace



Objects in Memory

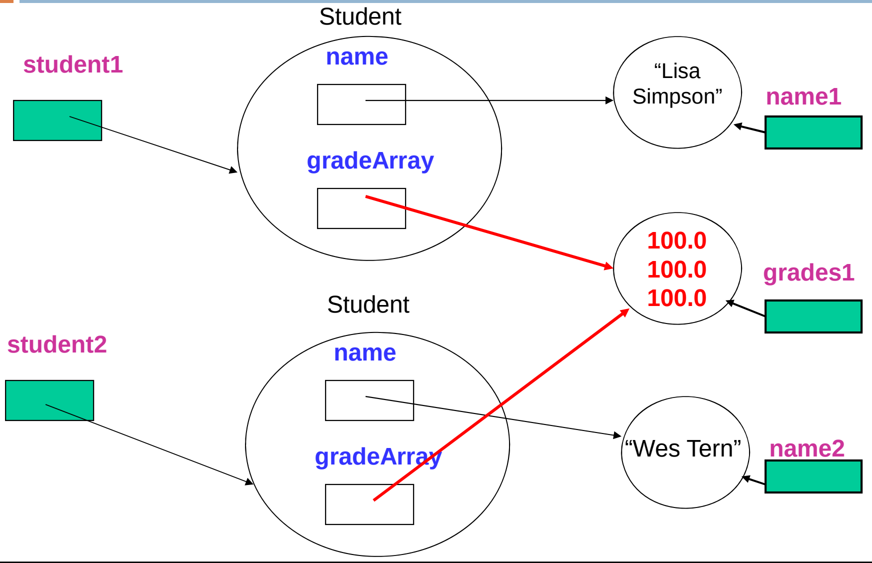
- The gradeArray variable for student1 and student2 both point to the same object
- Now consider the following method defined in the Student class, to change a student's grades:

```
public void changeGrades(double [] newGradeArray){
    for (int i = 0; i < newGradeArray.length; i++)
            this.gradeArray[i] = newGradeArray[i];
}
```

- We call

```
double [] newGrades = {100.0,100.0,100.0};
student1.changeGrades(newGrades);
```

Objects in Memory: a Trace



Objects in Memory

Lisa and Wes now have the same grades even though the method changeGrades was called only for Lisa's grades!

Where was the problem?

How can it be fixed?

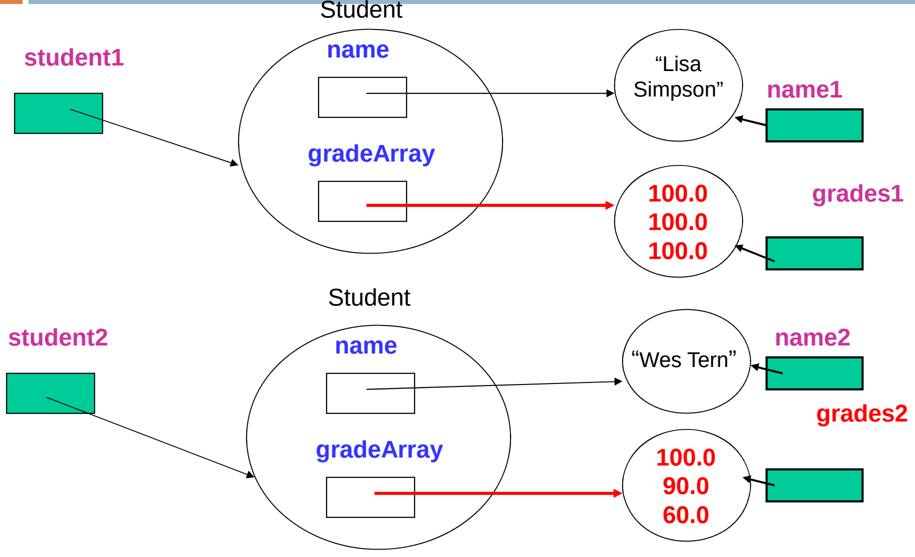
Objects in Memory

12

- Need a separate array object for "Wes Tern"
- Consider the following code segment instead

```
String name1 = "Lisa Simpson";
String name2 = "Wes Tern";
double [] grades1 = {100.0, 90.0,60.0};
double [] grades2 = {100.0, 90.0,60.0};
Student student1 = new Student(name1,grades1);
Student student2 = new Student(name2,grades2);
```

Objects in Memory: a Trace



Conclusion

- It is important to know which objects are being referenced by which reference variables.
- Diagrams help.

Passing Parameters

15

- You have been passing parameters to methods since you learned about the Turtle class
- Examples:

```
World world1 = new World();
Turtle turtle1 = new Turtle(100,200,world1);
turtle1.forward(50);
turtle1.turn(-90);
int howFar = 55;
turtle1.forward(howFar);
```

Formal and Actual Parameters

16

- Example: recall the method we defined for the Turtle class:

```
public void drawSquare(int width){
...
}
```

- The variable in the parameter list in the method definition is known as a formal parameter
- When we invoke a method with a parameter, that is known as an actual parameter, for example:

```
    turtle1.drawSquare(5);
```

- When the drawSquare method is executed, the value of the actual parameter size is copied to the formal parameter width

Passing Parameters: How it Works

```
public class TurtleDraw {
{
  public static void main(String[] args)
  {
    int size = 50;
    World world1 = new World();
    Turtle turtle1 = new Turtle(world1);
     turtle1.drawSquare(size);
    ….
```

actual parameter
is provided by the calling program when it invokes the method

```
public class Turtle  …   {
   …
  public void drawSquare(int width)
     {
       …
     }
}
```

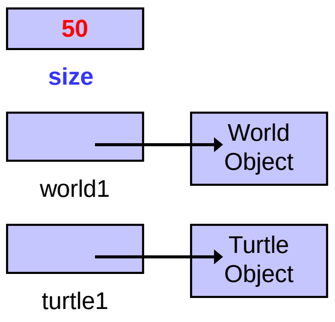
formal parameter
is part of the method definition

When the drawSquare method is executed, the value of the actual parameter size is ***copied to*** the formal parameter width

Passing Parameters: a Trace

In the calling program:

```
int size = 50;
World world1 = new World();
Turtle turtle1 = new Turtle(w);
```

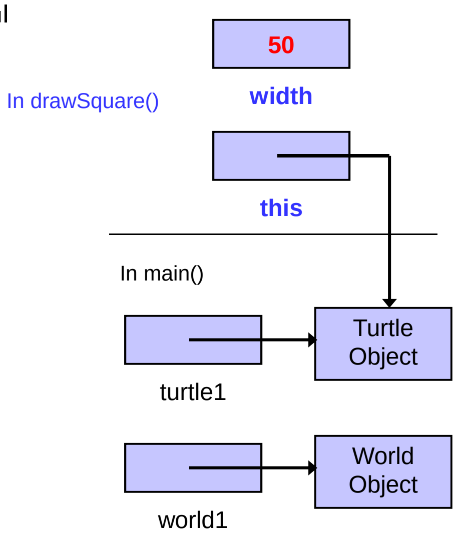


```
turtle1.drawSquare(size);
```

Passing Parameters: a Trace

In drawSquare: The value of the actual parameter size was copied to the parameter variable width

```
public void drawSquare(int width)
 {
   this.turnRight();
   this.forward(width);
   this.turnRight();
   this.forward(width);
   this.turnRight();
   this.forward(width);
   this.turnRight();
   this.forward(width);
 }
```



Passing Parameters

- Remember: the value of an actual parameter is copied *to* the parameter variable when the method starts executing
- In our example: the method drawSquare does not have any access to the actual variable size in the main method

Reference Variables as Parameters

- What happens when parameters are reference variables?
- We have seen several examples in the Picture class:

```
public void copyPicture (Picture pic)
public void copyPictureTo(Picture pic, int xStart, int yStart)
```

- What are the formal parameters in each of these method definitions?

Reference Variables as Parameters

- Consider a program containing the following code segment:

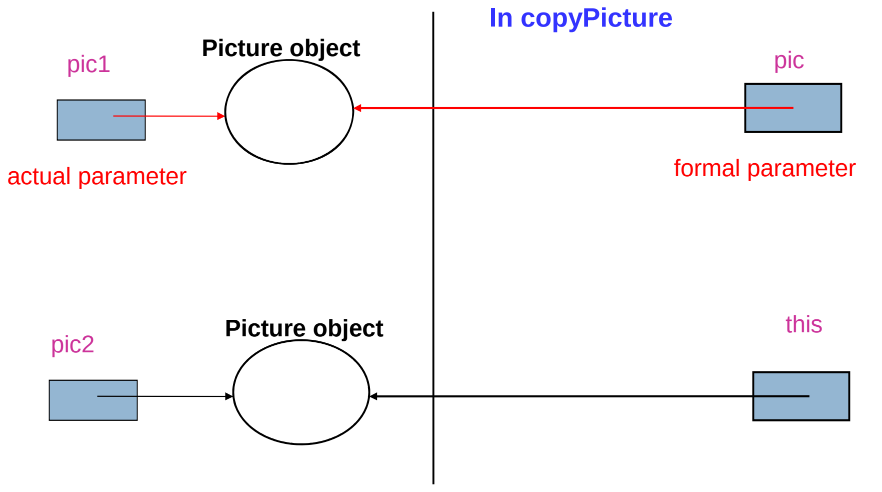
```
Picture pic1 = new Picture(…);
Picture pic2 = new Picture(…);
pic2. copyPicture(pic1);
```

- What happens when the copyPicture method is executed?

  pic1 is the actual parameter

Reference Variables as Parameters

23



Reference Variables as Parameters

24

- Recall that a reference variable "points to" an object, i.e. it contains the location of the object
- In Java, the value of an actual parameter is copied to the formal parameter when the method starts executing
  - The contents of actual parameter pic1 is copied to the formal parameter pic
  - So pic now contains the location of (points to) the same object as pic1

Reference Variables as Parameters

25

- As with parameters of primitive types, a method cannot change the contents of the actual parameter
- In our example: the method copyPicture does not have access to the actual variable pic1 in the main method
- However, it can change the data in the object that the actual parameter points to
  - Why? Because the formal parameter variable points to the same object
  - This is called a side-effect
  - It's usually safer to avoid them

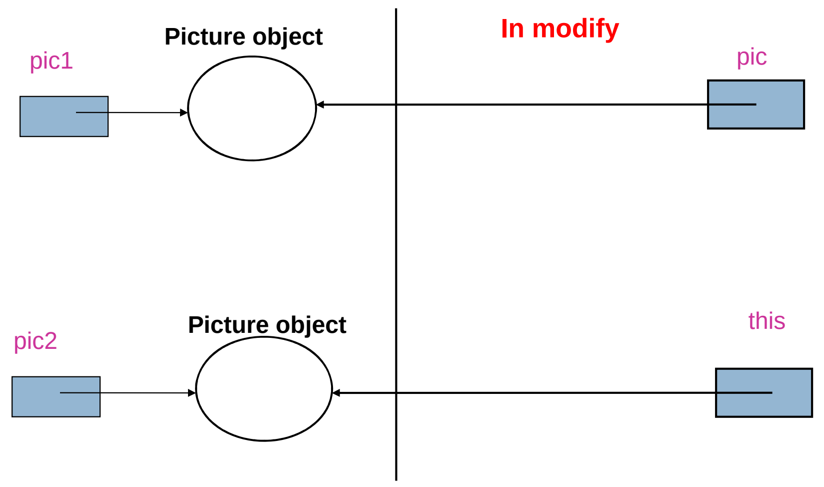
A side effect

26

```
public void modify(Picture pic)
{
  for (int x=0; x<pic.getWidth(); x++)
   for (int y=0; y<pic.getHeight(); y++)
   {
      Pixel pix1 = pic.getPixel(x, y);
      Pixel pix2 = this.getPixel(x, y);
      pix1.setColor(pix2.getColor());
    }
}
```

Reference variables as parameters

27



Summary

28

- How objects are stored in memory
  - Reference variables
  - Tracing
- How parameters are passed to methods