OBJECTS AND REFERENCES

Ch5.3

Objects and References: Outline

- Variables of a Class Type
- Defining an equals Method for a Class
- Boolean-Valued Methods
- Parameters of a Class Type

Variables of a Class Type

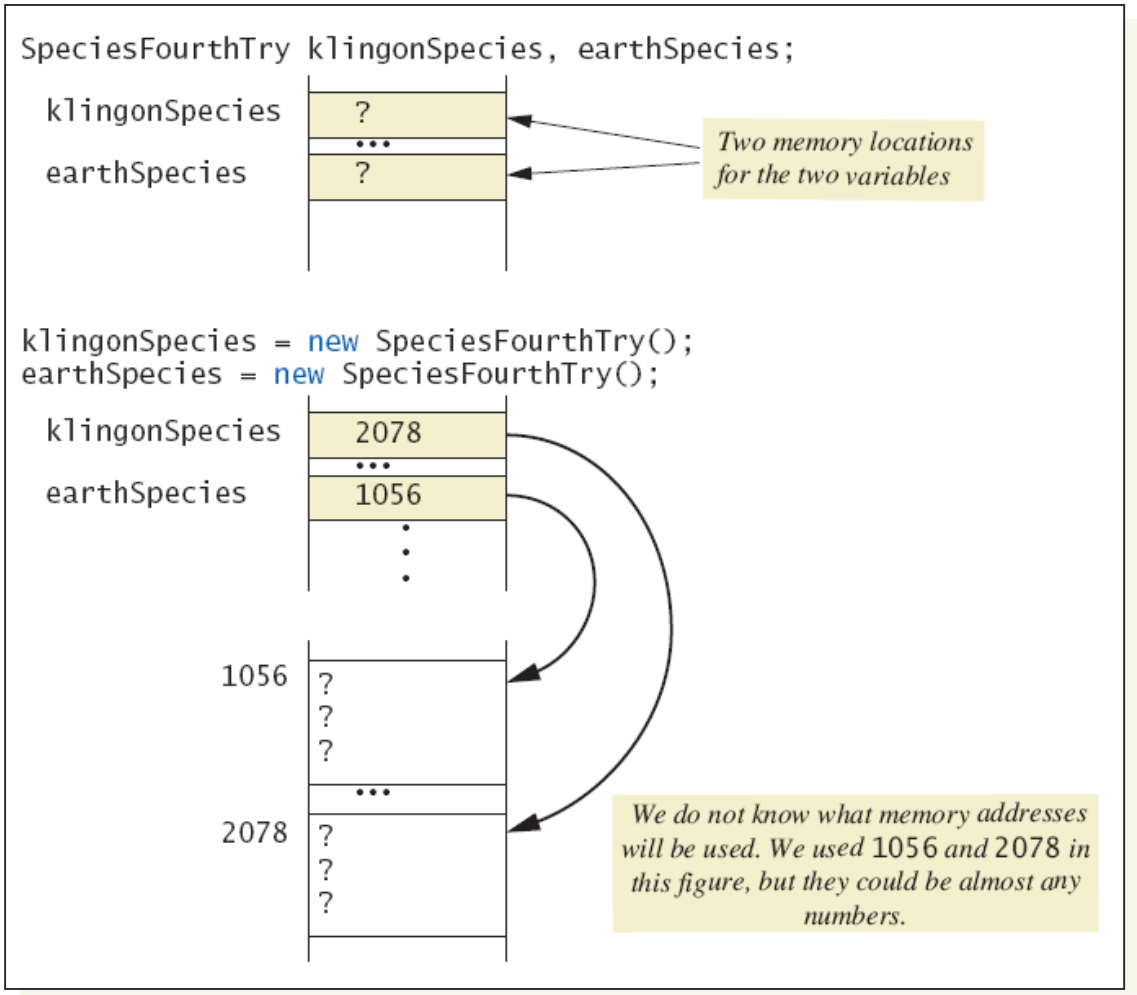
- All variables are implemented as a memory location
- Data of *primitive type* stored in the memory location assigned to the variable
- Variable of *class type* contains memory address of object named by the variable

Variables of a Class Type

- Object itself not stored in the variable
  - Stored elsewhere in memory
  - Variable contains address of where it is stored
- Address called the *reference* to the variable
- A *reference type* variable holds references (memory addresses)
  - This makes memory management of class types more efficient

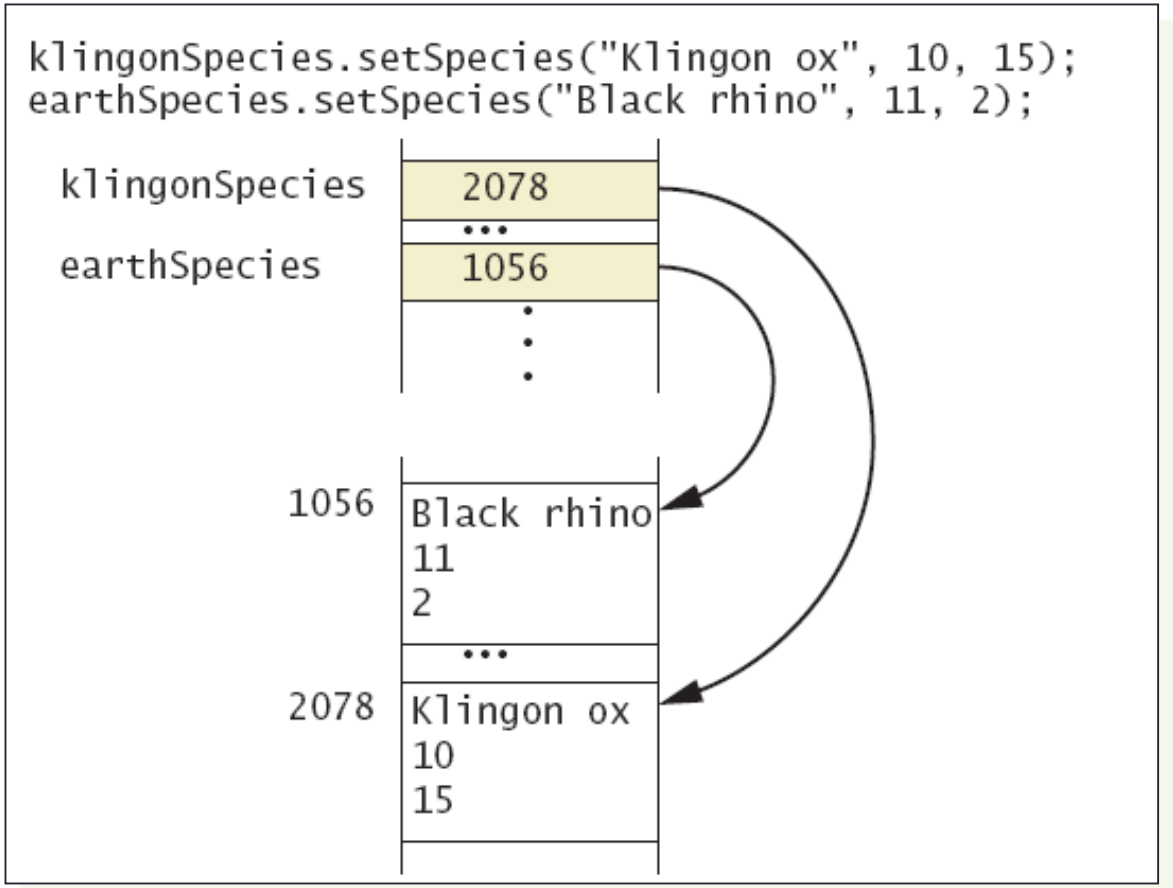
Variables of a Class Type

- Figure 5.5a Behavior of class variables



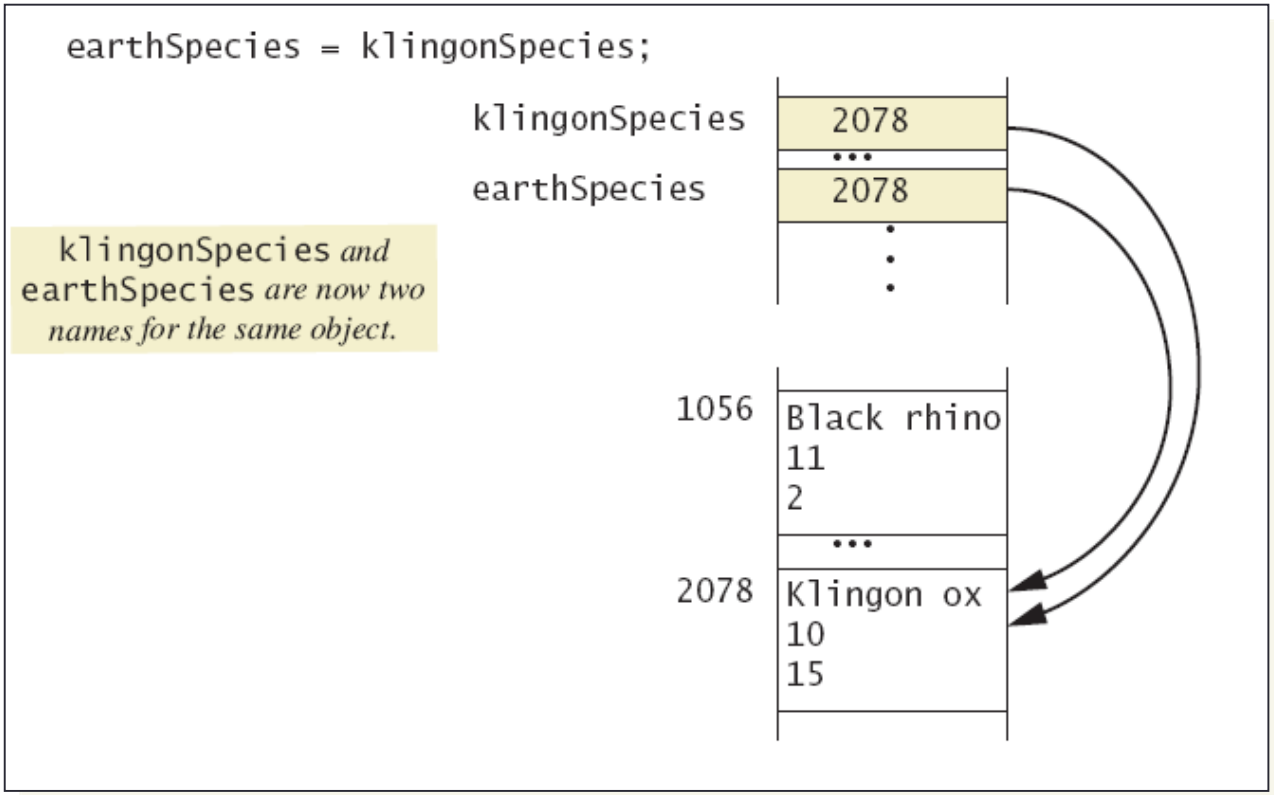
Variables of a Class Type

- Figure 5.5b Behavior of class variables



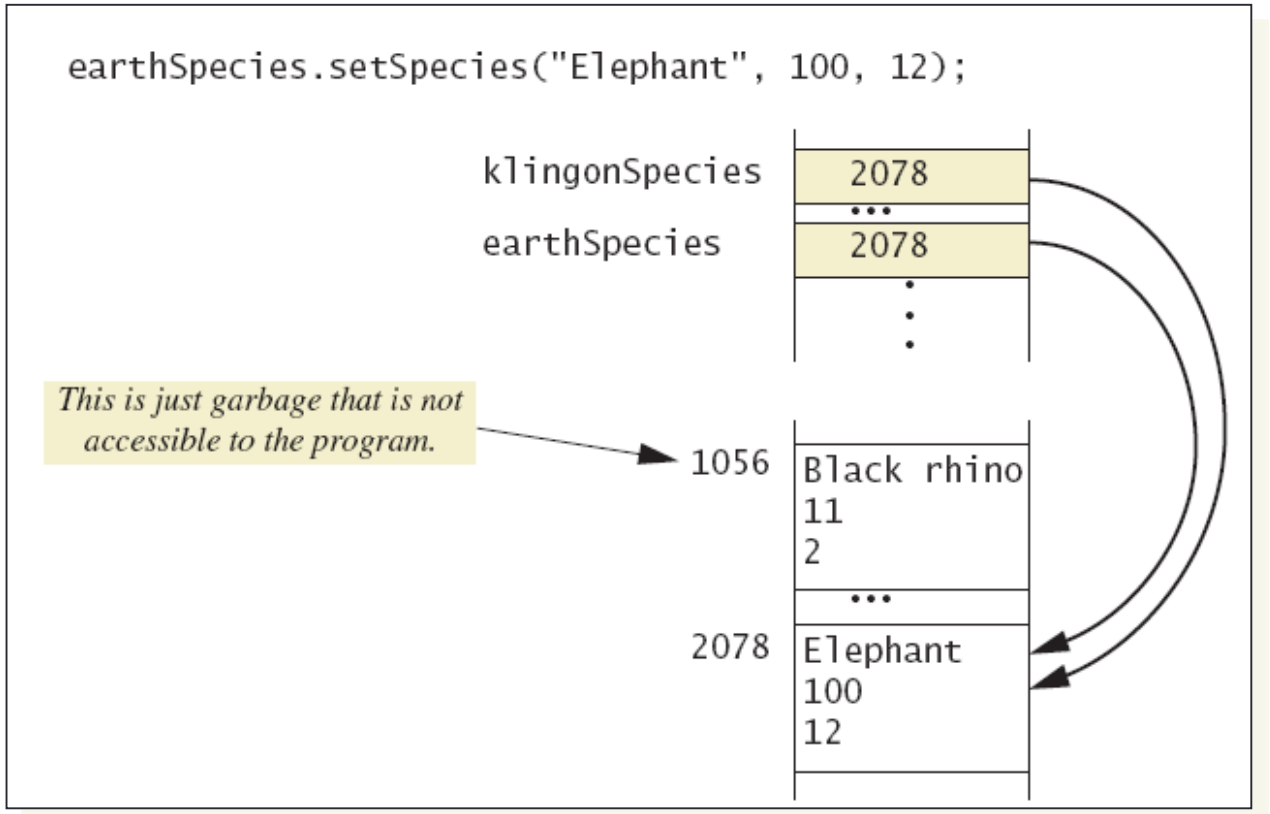
Variables of a Class Type

- Figure 5.5c Behavior of class variables



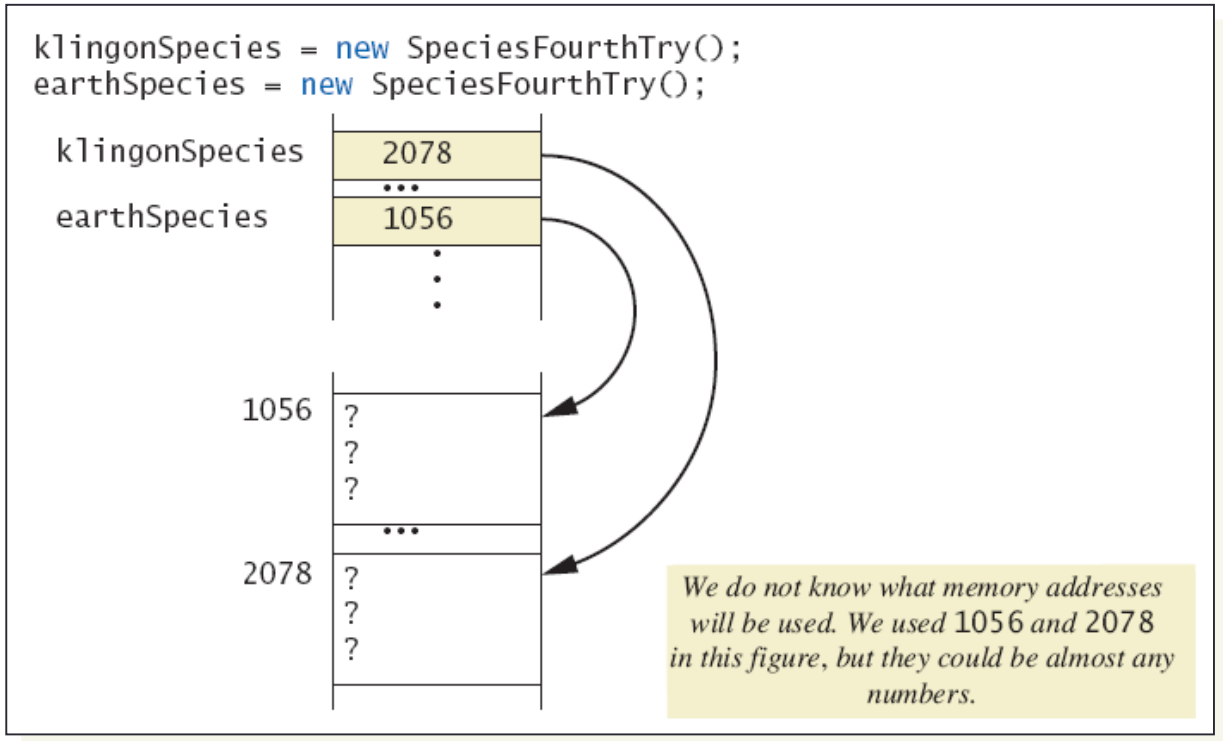
Variables of a Class Type

- Figure 5.5d Behavior of class variables



Variables of a Class Type

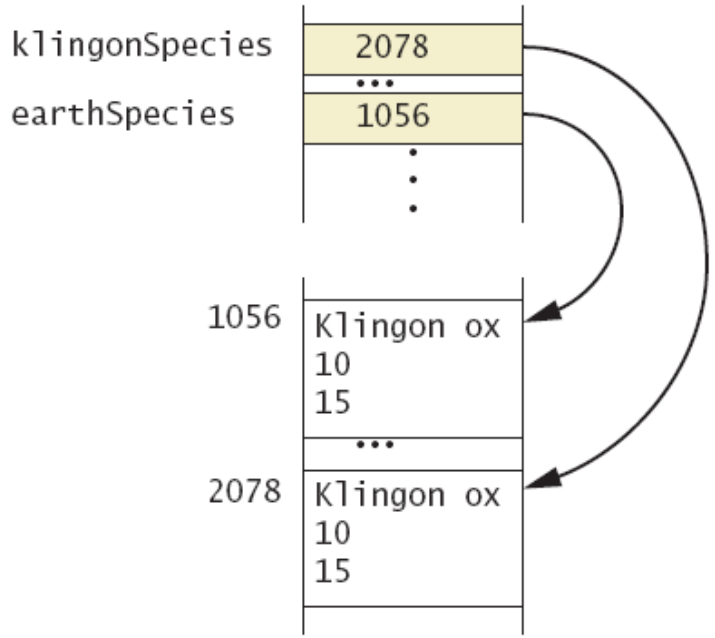
- Figure 5.6a Dangers of using == with objects



Variables of a Class Type

- Figure 5.6b Dangers of using == with objects

```
klingonSpecies.setSpecies("Klingon ox", 10, 15);
earthSpecies.setSpecies("Klingon ox", 10, 15);
```




```
if (klingonSpecies == earthSpecies)
    System.out.println("They are EQUAL.");
else
    System.out.println("They are NOT equal.");
```

The output is `They are Not equal` *, because* 2078 *is not equal to* 1056.

Defining an `equals` Method

- As demonstrated by previous figures
  - We cannot use == to compare two objects
  - We must write a method for a given class which will make the comparison as needed
- The `equals` for this class method used same way as `equals` method for `String`

Demonstrating an `equals` Method

```
public class Rectangle {
    private int width;
    private int height;

    public void setWidth(int w){  width = w; }
    public void setHeight(int h){ height = h; }

    public int getWidth() { return width; }
    public int getHeight() { return height; }

    public boolean equals(Rectangle r){
        return r.width == this.width &&
               r.height == this.height;
    }
}
```

Demonstrating an `equals` Method

```
public class RectangleTest {
    public static void main(String[] args) {
    Rectangle box1 = new Rectangle();
    Rectangle box2 = new Rectangle();
    box1.setWidth(5); box1.setHeight(10);
    box2.setWidth(5); box2.setHeight(10);
    if(box1 == box2)
        System.out.println("Match with ==.");
    else
        System.out.println("Do not match with ==.");

    if(box1.equals(box2))
        System.out.println("Match with the method equals.");
    else
        System.out.println("Do not match with the method equals");
    }
}
```

Note difference in the two comparison methods `==` versus `.equals( )`

Sample screen output

```
Do not match with ==.
Match with the method equals.
```

Object1 = object2

```
public class RectangleTest {
    public static void main(String[] args) {
    Rectangle box1 = new Rectangle();
    Rectangle box2 = new Rectangle();
    box1.setWidth(5); box1.setHeight(10);
    box2.setWidth(20); box2.setHeight(30);

    box1 = box2;
    System.out.println("The width of box 1 is " + box1.getWidth());
    System.out.println("The hight of box 1 is " + box1.getHeight());
    }
}
```

Sample screen output

```
The width of box 1 is 20
The height of box 1 is 30
```

Boolean-Valued Methods

- Methods can return a value of type **`boolean`**
- Use a **`boolean`** value in the **`return`** statement.

```
public boolean isPositive(int n) {
      return n > 0;
}
```

Parameters of a Class Type

- When assignment operator used with objects of class type
  - Only memory address is copied
- Similar to use of parameter of class type
  - Memory address of actual parameter passed to formal parameter
  - Formal parameter may access public elements of the class
  - Actual parameter thus can be changed by class methods

Parameters of Primitive type vs. class type

- Parameter of primitive type initialized with value of actual parameter
- Value of actual parameter not altered by method
- Parameter of class type initialized with address of actual parameter object
- Value of actual parameter may be altered by method calls

Parameter of primitive type vs. parameter of class Type (Example)

```
public class ParameterDemo {
    private int x;
    private int y;
    public void setValues(int x, int y){
        this.x = x;
        this.y = y;
    }
    public void tryToChange(int var1){
        var1 = x;}
    public void tryToReplace(ParameterDemo otherObject){
        otherObject = this;
    }
    public void change(ParameterDemo otherObject){
        otherObject.x = this.x;
        otherObject.y = this.y;
    }
    public void display(){
        System.out.println("x = "+ x + ", y = "+ y);} }
```

Parameter of primitive type vs. parameter of class Type (Example)

```
public class ParametersDemoTest {
    public static void main(String[] args) {
        ParameterDemo p1 = new ParameterDemo();
        ParameterDemo p2 = new ParameterDemo();
        p1.setValues(2, 3);
        p2.setValues(7, 9);

        int number = 20;
        System.out.println("number before calling tryToChange is " + number);
        p1.tryToChange(number);
        System.out.println("number after calling tryToChange is " + number);

        System.out.println("p2 before calling tryToReplace is ");
        p2.display();
        p1.tryToReplace(p2);
        System.out.println("p2 after calling tryToReplace is ");
        p2.display();

        p1.change(p2);
        System.out.println("p2 after calling change is ");
        p2.display();}}
```

Parameter of primitive type vs. parameter of class Type (Example)

Sample screen output

```
number before calling tryToChange is 20
number after calling tryToChange is 20
p2 before calling tryToReplace is
x = 7, y = 9
p2 after calling tryToReplace is
x = 7, y = 9
p2 after calling change is
x = 2, y = 3
```

Summary

- Classes have
  - Instance variables to store data
  - Method definitions to perform actions
- Instance variables should be private
- Class needs accessor, mutator methods
- Methods may be
  - Value returning methods
  - Void methods that do not return a value

Summary

- Keyword this used within method definition represents invoking object
- Local variables defined within method definition
- Formal arguments must match actual parameters with respect to number, order, and data type
- Formal parameters act like local variables

Summary

- Parameter of primitive type initialized with value of actual parameter
  - Value of actual parameter not altered by method
- Parameter of class type initialized with address of actual parameter object
  - Value of actual parameter may be altered by method calls
- A method definition can include call to another method in same or different class