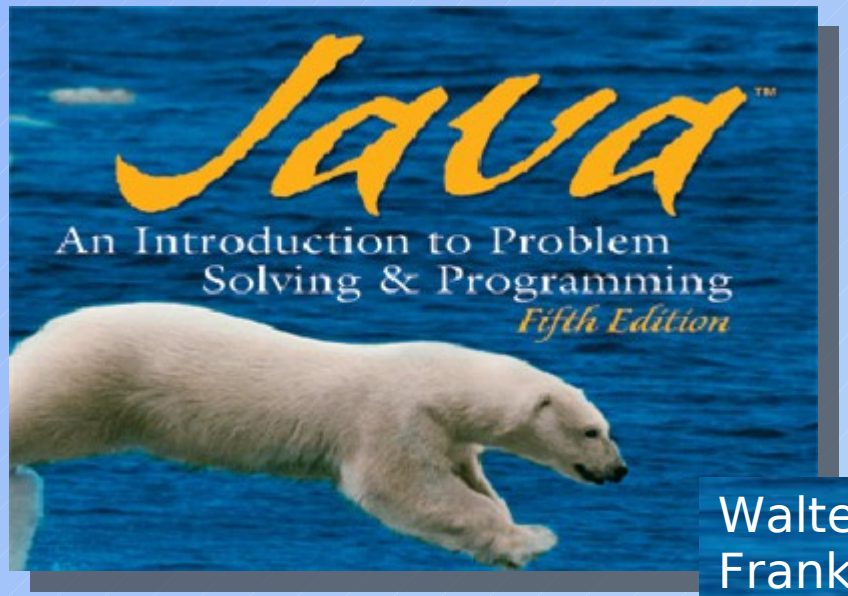


Walter Savitch
Frank M. Carrano

Defining Classes and Methods

Chapter 5b

*JAVA: An Introduction to Problem Solving & Programming, 5*th Ed. By Walter Savitch and Frank Carrano.
ISBN 0136130887

Information Hiding

- Programmers using a class method need not know details of implementation
  - Only needs to know *what* the method does
- Information hiding:
  - Designing a method so it can be used without knowing details
- Also referred to as *abstraction*
- Method design should separate *what* from *how*

The `public` and `private` Modifiers

- Type specified as **`public`**
  - Any other class can directly access that object by name
- Classes generally specified as **`public`**
- Instance variables usually not **`public`**
  - Instead specify as **`private`**

Programming Example

- Demonstration of need for private variables
- Download **`Rectangle`**
- Statement such as

  `box.width = 6;`

  is illegal since width is **`private`**
  - Keeps remaining elements of the class consistent in this example

Programming Example

- Another implementation of a Rectangle class
- Download **`Rectangle2`**
- Note **`setDimensions`** method
  - This is the only way the **`width`** and **`height`** may be altered outside the class

Accessor and Mutator Methods

- When instance variables are private the class must provide methods to access values stored there
  - Typically named **`getSomeValue`**
  - Referred to as an accessor method
- Must also provide methods to change the values of the private instance variable
  - Typically named **`setSomeValue`**
  - Referred to as a mutator method

Accessor and Mutator Methods

- Consider an example class with accessor and mutator methods
- Download **`SpeciesFourthTry`** and **`SpeciesFourthTryDemo`**
- Note the mutator method
  - **`setSpecies`**
- Note accessor methods
  - **`getName`**, **`getPopulation`**, **`getGrowthRate`**

Methods Calling Methods

- A method body may call any other method
- If the invoked method is within the same class
  - Need not use prefix of receiving object
- Download **`Oracle`** and **`OracleDemo`**

Encapsulation

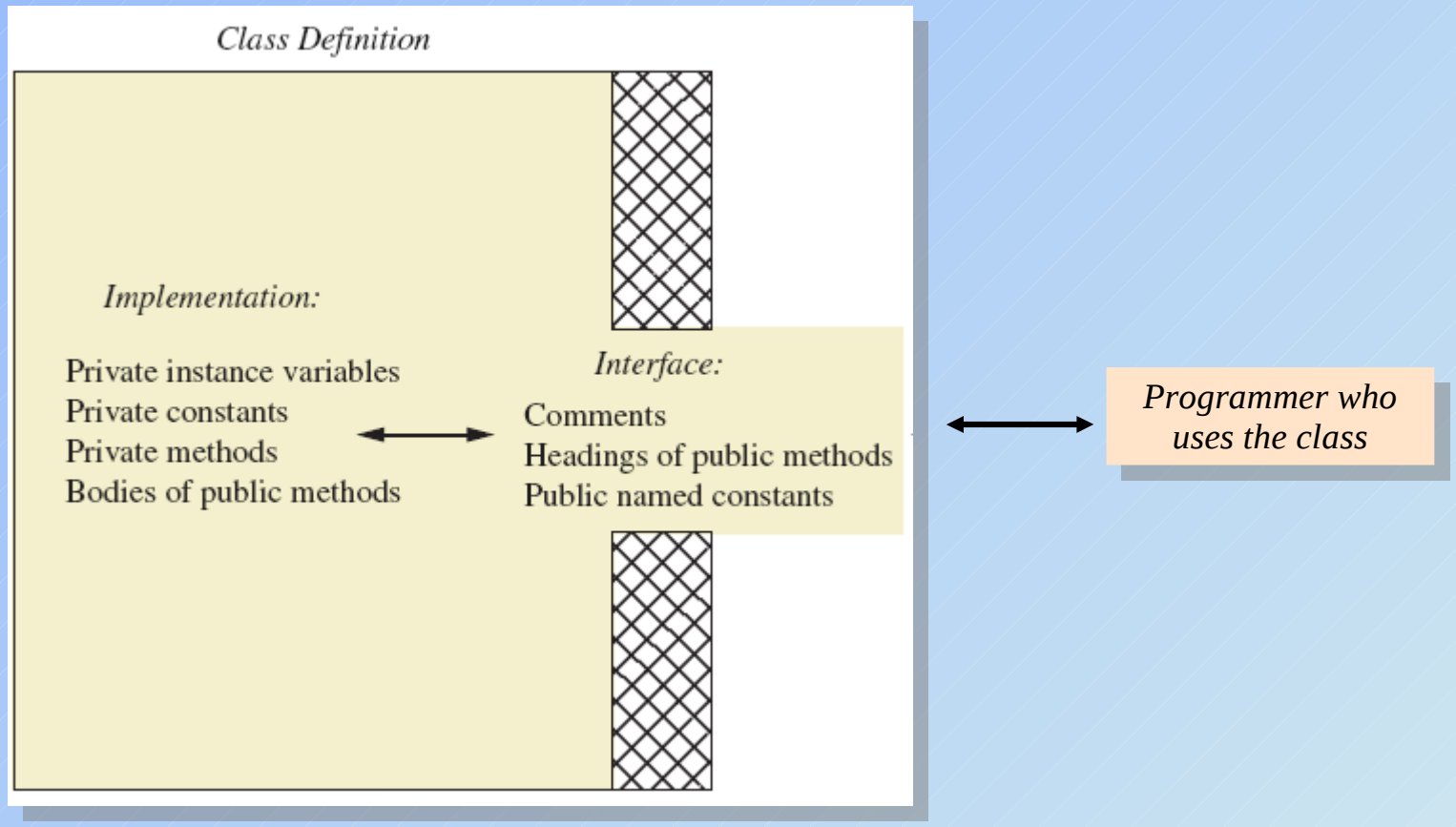
- Consider example of driving a car
  - We see and use break pedal, accelerator pedal, steering wheel – know what they do
  - We do not see mechanical details of how they do their jobs
- Encapsulation divides class definition into
  - Class interface
  - Class implementation

Encapsulation

- A *class interface*
  - Tells what the class does
  - Gives headings for public methods and comments about them
- A *class implementation*
  - Contains private variables
  - Includes definitions of public and private methods

Encapsulation

- Figure 5.3 A well encapsulated class definition



Encapsulation

- Preface a class definition with comment on how to use class.
- Declare all instance variables in the class as private.
- Provide public accessor methods to retrieve data.
- Provide public methods manipulating data.
  - Such methods could include public mutator methods.
- Place a comment before each public method heading that fully specifies how to use the method.
- Make any helping methods private.
- Write comments within class definition to describe implementation details.

Automatic Documentation `javadoc`

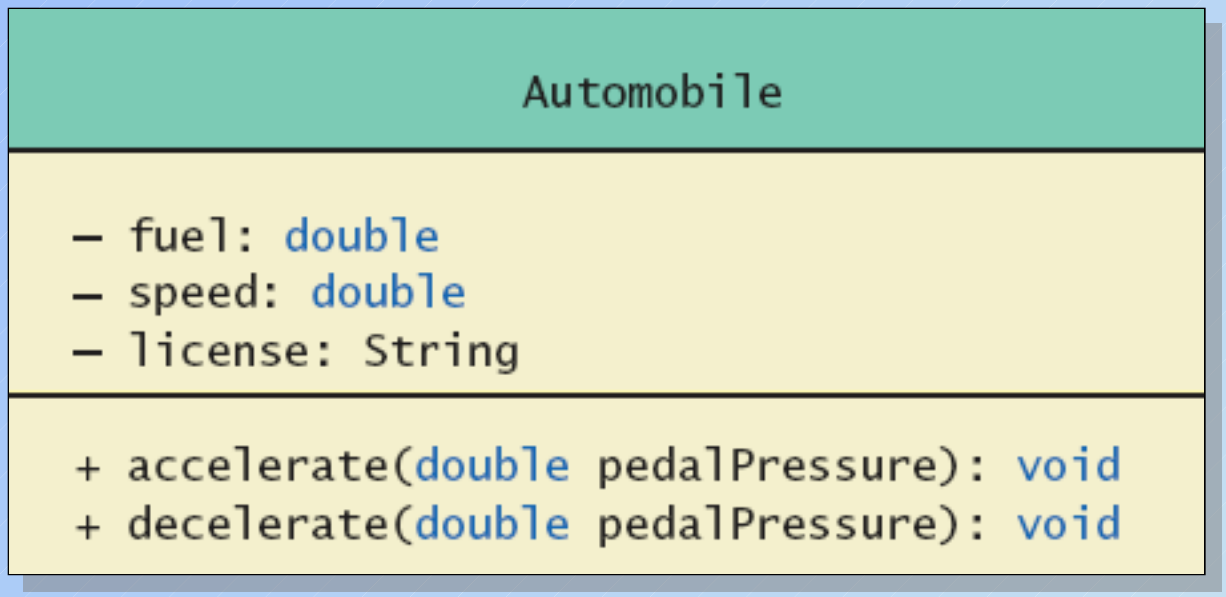
- Generates documentation for class interface
- Comments in source code must be enclosed in `/**  */`
- Utility **`javadoc`** will include
  - These comments
  - Headings of public methods
- Output of **`javadoc`** is HTML format

Automatic Documentation `javadoc`

- Add `javadoc` comments to the `Rectangle` class
- In drjava
  - Tools -> Javadoc -> Preview Javadoc for Current Document
  - May have to set browser first:
    - Edit -> Preferences -> Resource Locations
    - Enter browser command (firefox,...)

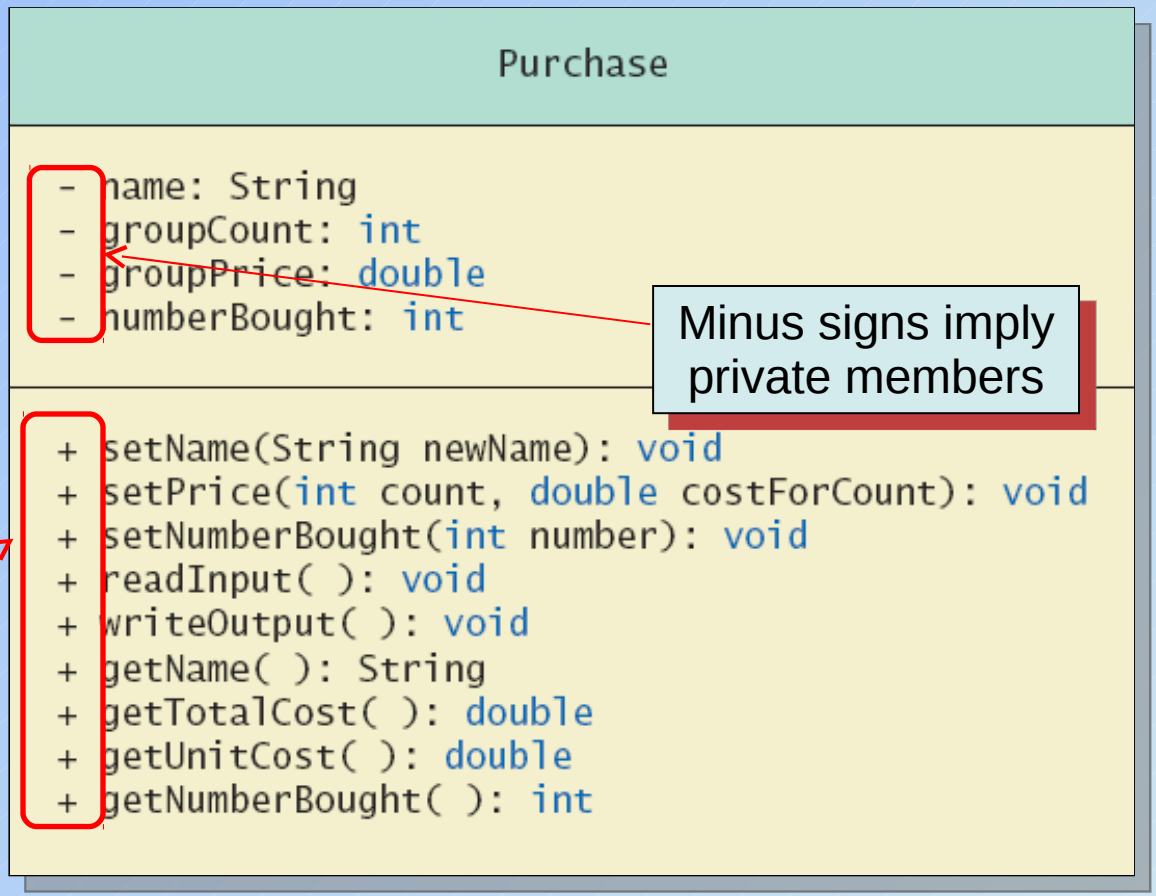
UML Class Diagrams

- Recall Figure 5.2, a class outline as a UML class diagram



UML Class Diagrams

- UML for a **Purchase** class



UML Class Diagrams

- Contains more than interface, less than full implementation
- Usually written *before* class is defined
- Used by the programmer defining the class
  - Contrast with the interface used by programmer who uses the class

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

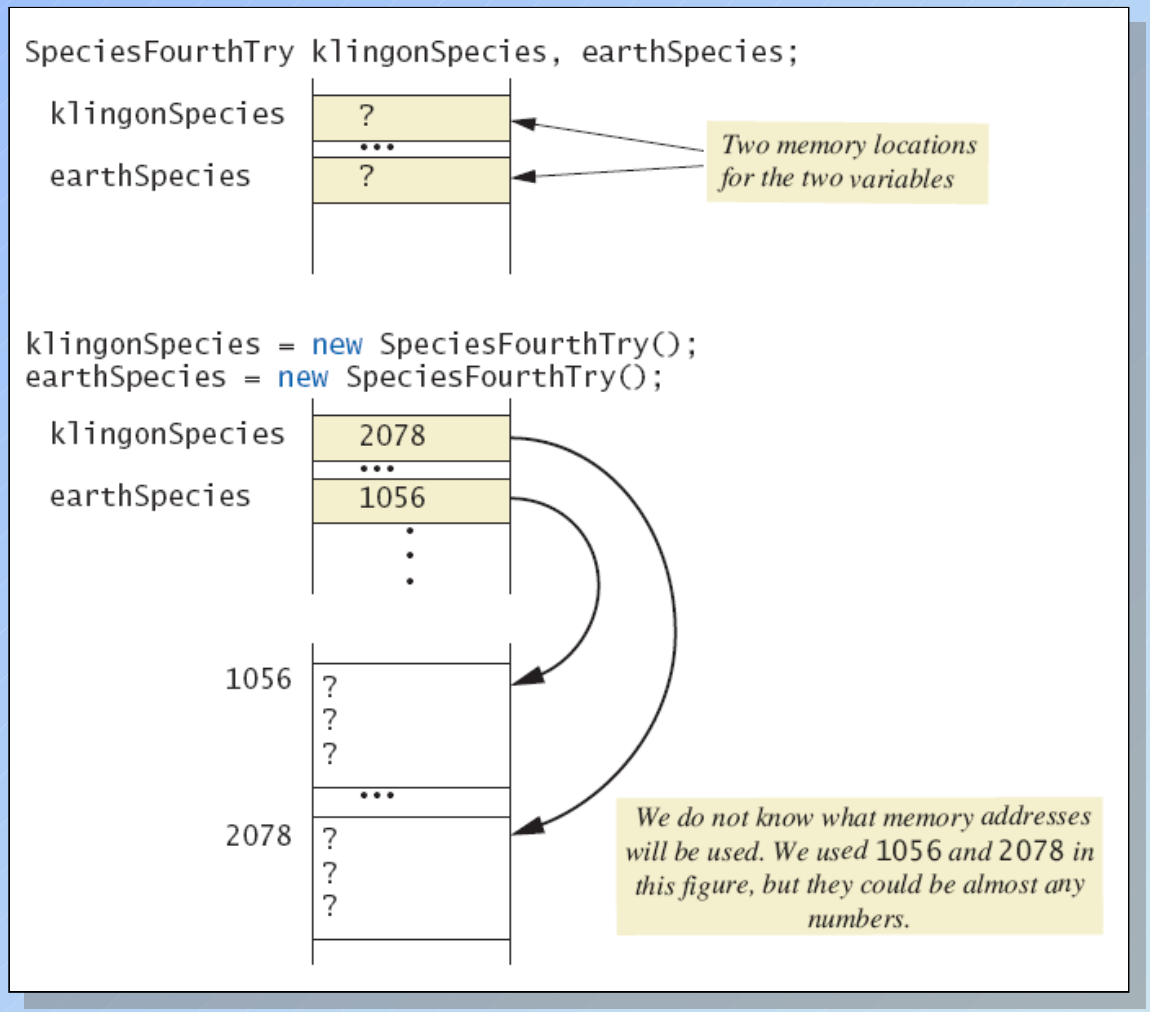
- All variables are implemented as a memory location
- Data of *primitive type* stored in the memory location assigned to the variable
- Variable of *class type* contains memory address of object named by the variable

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

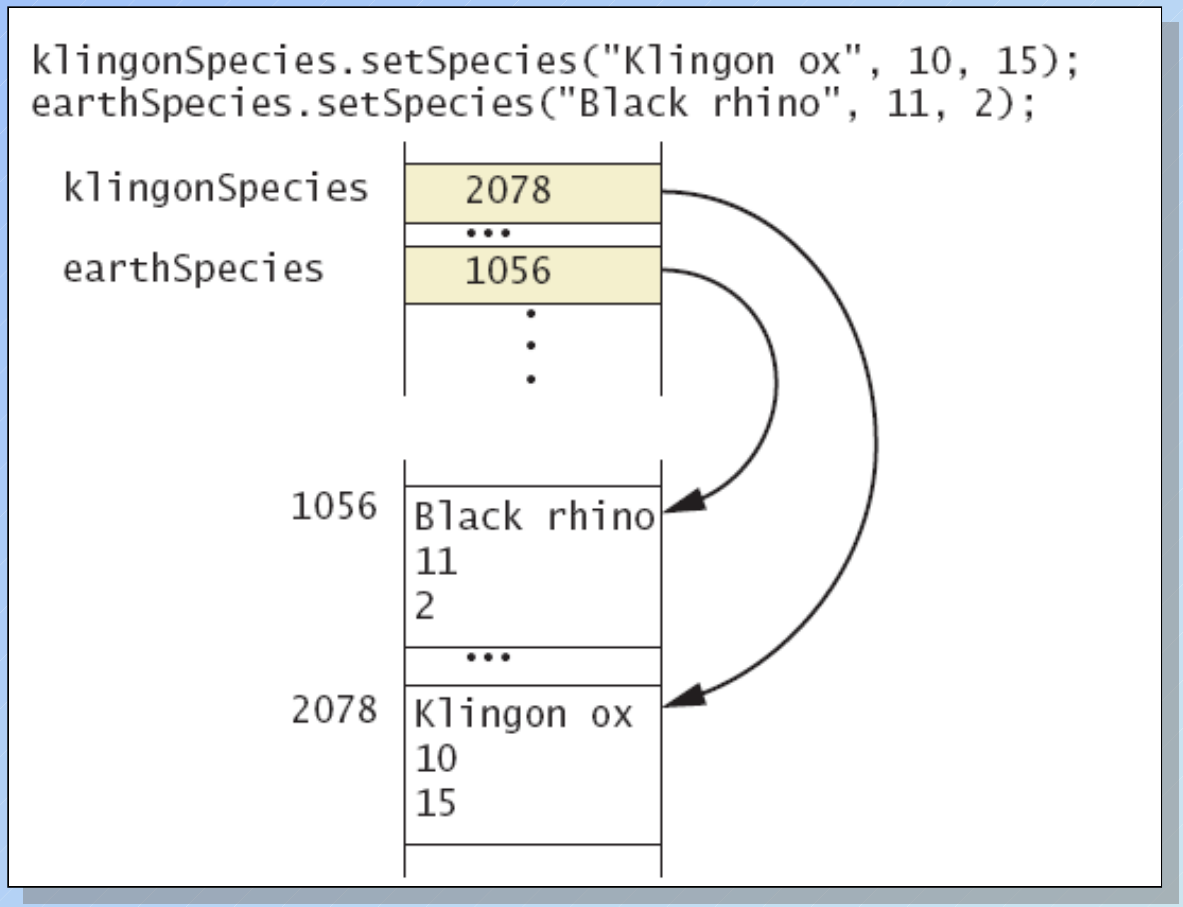
Variables of a Class Type

- Behavior of class variables



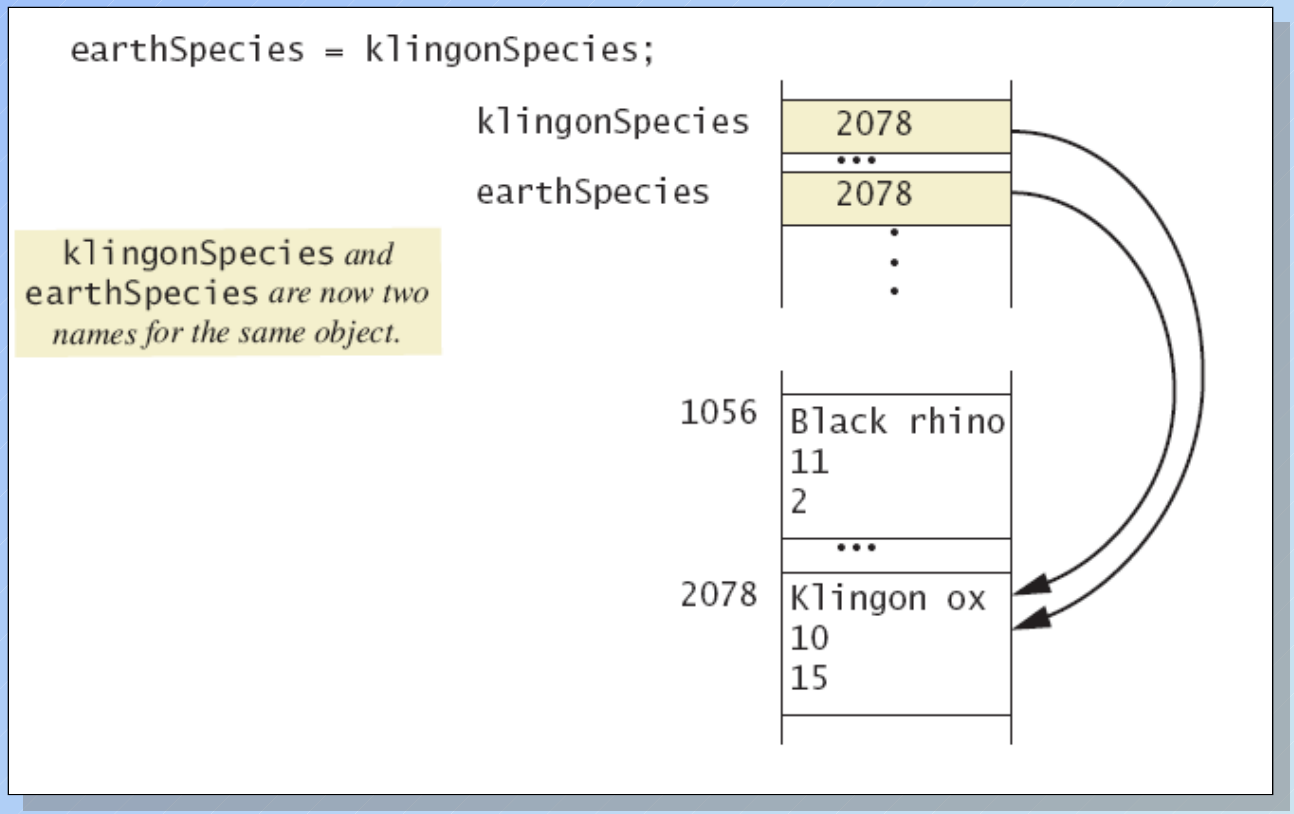
Variables of a Class Type

- Behavior of class variables



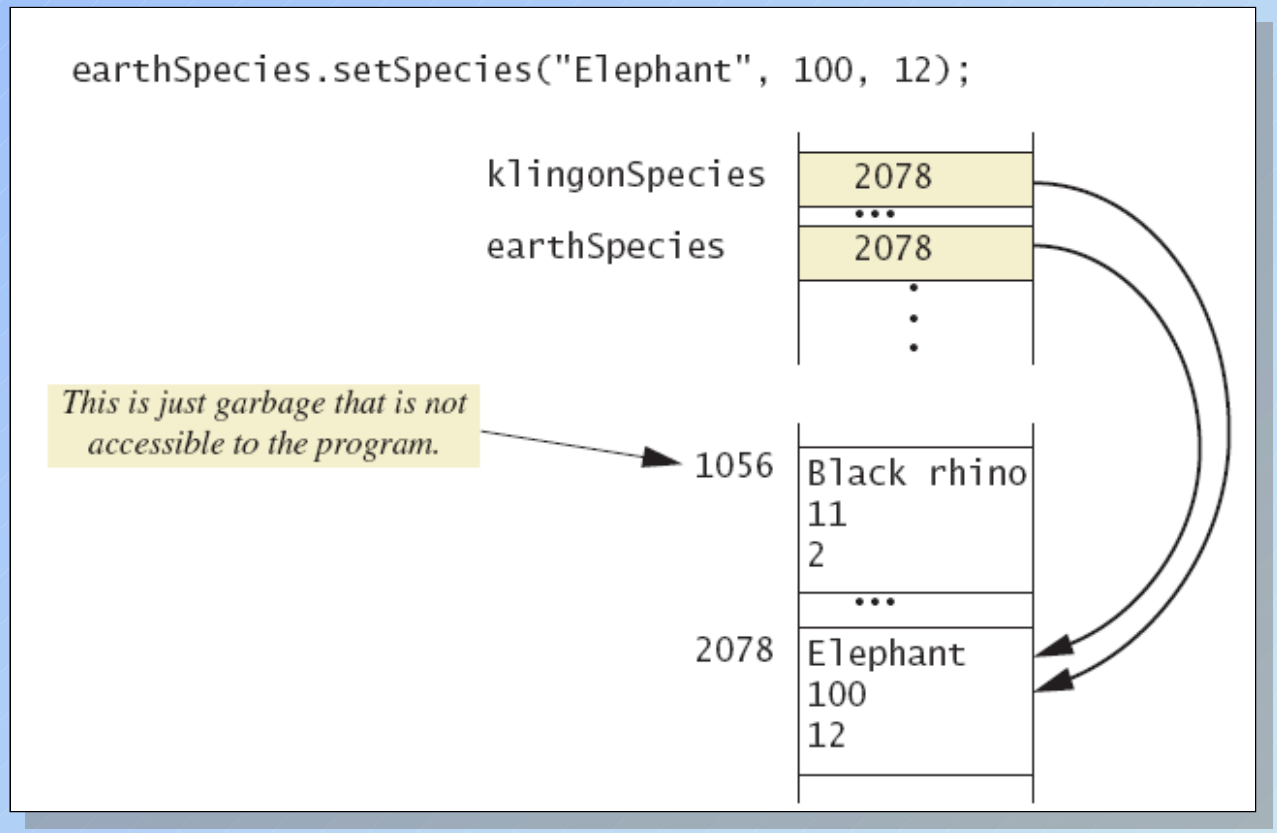
Variables of a Class Type

- Behavior of class variables



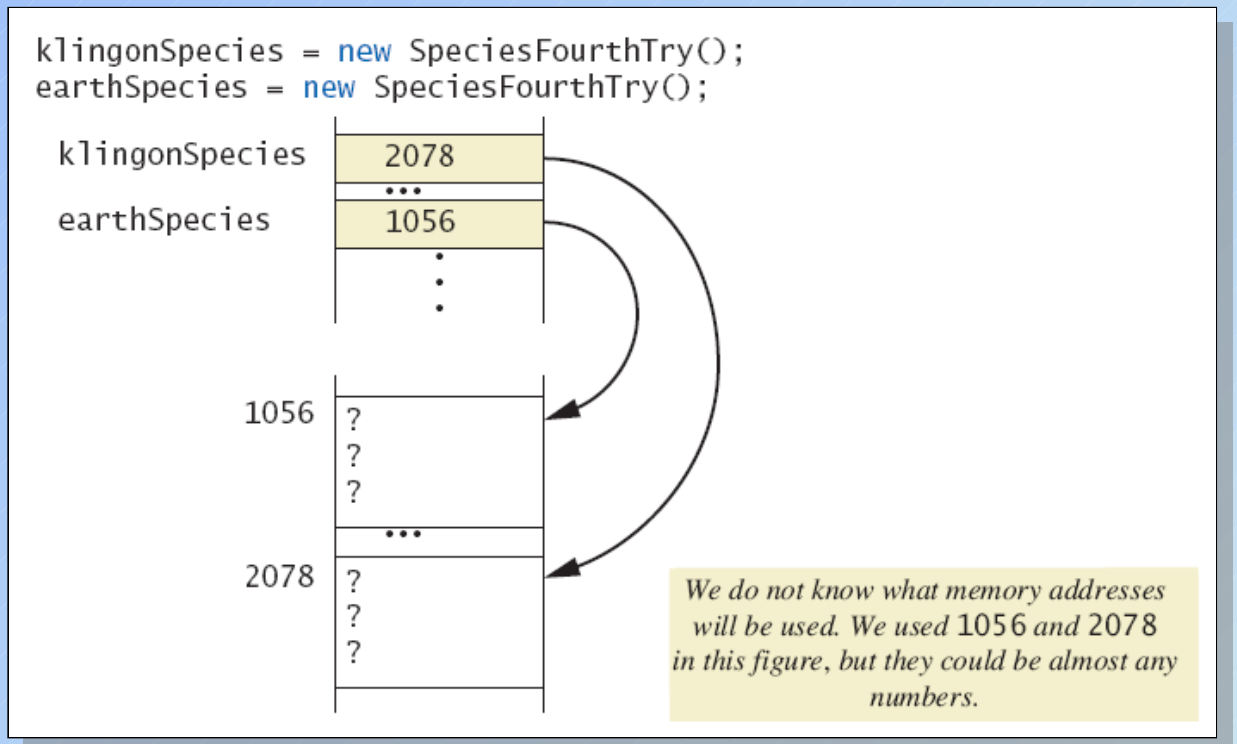
Variables of a Class Type

- Behavior of class variables



Variables of a Class Type

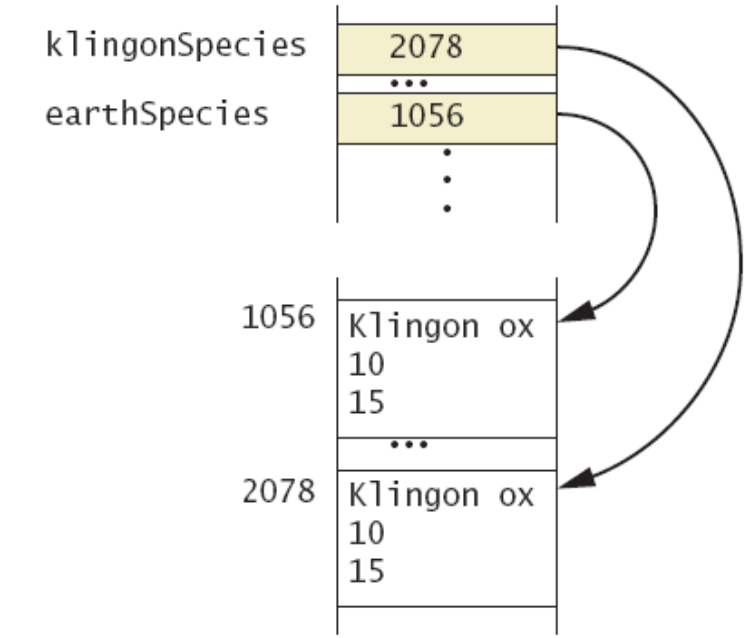
- Dangers of using == with objects



Variables of a Class Type

- 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
- Download **`Species`** and **`SpeciesEqualsDemo`**
- The **`equals`** for this class method used same way as **`equals`** method for **`String`**

Boolean-Valued Methods

- Methods can return a value of type **`boolean`**
- Use a **`boolean`** value in the **`return`** statement
- Add this method to the **`Species`** class

```
/**
 Precondition: This object and the argument otherSpecies
 both have values for their population.
 Returns true if the population of this object is greater
 than the population of otherSpecies; otherwise, returns false.
*/
public boolean isPopulationLargerThan(Species otherSpecies)
{
    return population > otherSpecies.population;
}
```

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

Programming Example

- Download **`DemoSpecies`**
  - Note different parameter types and results
- Download **`ParametersDemo`**
  - Parameters of a class type versus parameters of a primitive type