Defining Classes and Methods

Chapter 5
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

```java
klingonSpecies.setSpecies("Klingon ox", 10, 15);
earthSpecies.setSpecies("Black rhino", 11, 2);
```
Variables of a Class Type

• Figure 5.5c

Behavior of class variables

![Diagram showing variables of a class type](image)

- earthSpecies = klingonSpecies;
- klingonSpecies and earthSpecies are now two names for the same object.
Variables of a Class Type

- Figure 5.5d
- Behavior of class variables

```java
earthSpecies.setSpecies("Elephant", 100, 12);
```

This is just garbage that is not accessible to the program.
Variables of a Class Type

Figure 5.6a
Dangers of using `==` with objects

```java
klingonSpecies = new SpeciesFourthTry();
earthSpecies = new SpeciesFourthTry();
```

We do not know what memory addresses will be used. We used 1056 and 2078 in this figure, but they could be almost any numbers.
Variables of a Class Type

• Figure 5.6b
Dangers of using `==` with objects

```java
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

• View sample code, listing 5.17

```java
class Species
{
    // The equals method for this class method used same way as equals method for String
}
```
Demonstrating an `equals` Method

- View sample program, listing 5.18
  ```java
class SpeciesEqualsDemo
```
- Note difference in the two comparison methods `==` versus `.equals()`

Sample screen output:

Do Not match with `==`
Match with the method `equals`
Now we change one Klingon ox to all lowercase.
Match with the method `equals`
Complete Programming Example

• View sample code, listing 5.19 class **Species**
• Figure 5.7 Class Diagram for the class **Species** in listing 5.19

```java
public class Species {
    private String name;
    private int population;
    private double growthRate;

    public Species(String name, int population, double growthRate) {
        this.name = name;
        this.population = population;
        this.growthRate = growthRate;
    }

    public void readInput() {
        // read input
    }

    public void writeOutput() {
        // write output
    }

    public int predictPopulation(int years) {
        return population * (1 + growthRate) * Math.pow(1 + growthRate, years);
    }

    public void setSpecies(String newName, int newPopulation, double newGrowthRate) {
        name = newName;
        population = newPopulation;
        growthRate = newGrowthRate;
    }

    public String getName() {
        return name;
    }

    public int getPopulation() {
        return population;
    }

    public double getGrowthRate() {
        return growthRate;
    }

    public boolean equals(Species otherObject) {
        return name.equals(otherObject.getName());
    }
}
```
Boolean-Valued Methods

• Methods can return a value of type `boolean`
• Use a `boolean` value in the `return` statement
• Note method from listing 5.19

```java
/**
   * 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

• View sample code, listing 5.21
  class DemoSpecies
  ▪ Note different parameter types and results

• View sample program, listing 5.22
  ▪ Parameters of a class type versus parameters of a primitive type
  class ParametersDemo
Programming Example

aPopulation BEFORE calling tryToChange: 42
aPopulation AFTER calling tryToChange: 42
s2 BEFORE calling tryToReplace:
Name = Ferengie Fur Ball
Population = 90
Growth Rate = 56.0%
s2 AFTER calling tryToReplace:
Name = Ferengie Fur Ball
Population = 90
Growth Rate = 56.0%
s2 AFTER calling change:
Name = Klingon ox
Population = 10
Growth Rate = 15.0%
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
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
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
Summary

• Utility program javadoc creates documentation
• Class designers use UML notation to describe classes
Summary

• Operators `==` and `===` behave differently with objects of class types (vs. primitive types)
• Designer of class should include an `equals` method