Object-Oriented Programming I

Variables In-Depth

Slides by Magdin Stoica
Updates by Georg Feil

Learning Outcomes

1. Analyze categorizes of variables used in object-oriented programs
2. Compare and contrast field variables with local variables in regards to meaning, accessibility, visibility, naming conventions, and use.
3. Explore the use and definition of static variables.
4. Explore the use and definition of constants in computer programs
5. Define the object-oriented principle of encapsulation (as applied to data) and its role in protecting an object's identity
6. Create programs that consist of more than one class

Reading Assignments

- Introduction to Java Programming (required)
  - Chapter 2: Elementary Programming, Section 2.7 only
  - Chapter 8: Objects and Classes, Section 8.9 only
  - Chapter 10: Thinking in Objects, Section 10.3 only

- Head First Java (recommended)
  - Chapter 4: Methods Use Instance Variables.
    Sections “Declaring and initializing instance variables” and “The difference between instance and local variables”

Variables and Scope

- Variables are always declared in a named scope: either a class scope or a method scope
- Variables declared directly in the class scope are called field variables or simply fields
  - Their accessibility from other parts of the program, from other scopes depends on the visibility modifier
- Variables declared inside method scopes are called local variables
  - They are only visible inside the method scope
  - Accessible by statements that are part of the method scope which follow the variable declaration
  - Cannot be accessed outside the method that declares them
  - Are forgotten each time the method completes (returns). There is no memory of their value the next time the method is called!

Field Variables

- Field variables give objects their identity
  - Fields define the attributes of an object
- Recall our barking dogs example program…
  - Each dog has a name and a weight (can you think of some other attributes a dog may have?)
  - Each object instance has its own copy of all field variables (except those that use 'static')
- Think of fields as "long-term memory"
  - Once a value is given to a field it will stay part of the object for as long as the object exists

Field Variable Accessibility

- The accessibility of field variables is determined by their visibility property
  - private fields are only accessible from methods of the same class and NOT from the outside
  - public fields are accessible from any class
- If visibility is not specified, fields are accessible from all classes in the same package, not other packages
  - We won't use this in PROG10082!
  - Always use public or private for field variables (normally private)

Data Encapsulation

- Recall that we always prefer to make field variables private. Why?
- Lets us control access
  - Write a "getter" if its value needs to be accessed by other classes
  - Write a "setter" if its value needs to be changed by other classes
  - Don't write a getter/setter to keep the field private (hidden)
- This is called Data Encapsulation
  - Encapsulation is an important idea in Object-Oriented Programming
  - Idea is to allow access to data (variables) *only when necessary*
  - Leads to better designed programs, fewer bugs
  - The internal data storage format can change if needed without changing the interface (methods)

Default Initial Values for Fields

- Field variables are automatically initialized to zero whether an initialization statement is used or not
  - What does zero mean for different types?

Type	Default Value (zero)
Integral	0 (zero)
Floating Point	0.0
Boolean	false
Character	0 (character with the ASCII code zero), non-printable character
String	null (keyword that means “nothing”)
Any class	null

Local Variables

- Are defined inside a method scope or an unnamed scope inside of a method
- Think of local variables as "short-term memory"
  - A local variable only exists from the moment the statement that declares is executed until the scope it is defined in ends
- The visibility is fixed to the scope they are defined in
  - Local variables are only visible to statements inside the method which follow the variable declaration
  - Visibility modifiers cannot be used in local variables
  - If defined inside an un-named block inside a method block they are only visible inside the un-named block
- Must be initialized explicitly before they can be used
  - Attempting to use an uninitialized local variable leads to compile errors
  - Unlike fields, they are not automatically initialized to "zero"

Field Variables

Long-Term Memory

Local Variables

Short-Term Memory

Variables Naming Convention

- To differentiate between field variables and local variables we use a separate naming convention
- Field variables start with _ (underscore) followed by the usual lower-case letter
  - The _ is called a "prefix"
  - Emphasizes their importance in establishing an object's identity, its characteristics and the long-term memory aspects
- Local variables do not have any prefix and start with a lower case letter
  - W usually have more local variables than fields so it makes sense to use no prefix for the many and prefix for the few

Example: Fields vs. Local

Class declaration

Class definition

```
public class Person
{
    private String _firstName;
    private String _lastName;

    public String calculateFullName()

    {
        String fullName = _firstName + " " + _lastName;

        return fullName;

    }

    public String calculateFormalFullName()

    {
        String fullName = _lastName + ", " + _firstName;

        return fullName;

    }

}
```

Field variables (fields)

Local variable

Local variable

Example: Fields vs. Parameters

Class declaration

Class definition

```
public class Person
{
    private String _firstName;
    private String _lastName;


    public void setFirstName(String firstName)
    {
        _firstName = firstName;
    }
}
```

Field variables (fields)

Parameter (local variable initialized by the caller)

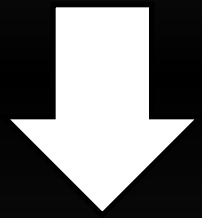
Summary Fields vs. Local Variables

Field Variables (Fields)	Local Variables
Long-term memory	Short-term memory
Form the object's identity	Simple "helper" variables
Class scope	Method scope
Visibility can be customized with visibility modifiers	Visibility is fixed to the statements in the same scope
Can be accessed if public or if accessor / mutator methods exist	Cannot be accessed from outside the scope. They are used by not accessed directly.
All methods of the class have access to these variables	Only the method that declares has access
Initialized by default to "zero" or null depending on the data type	Do not have an initial value. Must be explicitly initialized
Name are prefixed with _	Names have no prefix

null Variables

- Object variables which have not been created have the value "null"
  - Fields that are objects are by default null (e.g. String _name;)
  - Local object variables can be initialized to null (e.g. String name = null;)
- Null means "nothing", no object, object does not exist
- Null can only be used with object variables
  - Cannot initialize an int to "null"
  - Cannot initialize a double to "null"
  - Cannot initialize a boolean to "null"
  - Cannot initialize a char to "null"
- Strings are objects too, string fields are by default null
  - private String name; // What is the value of "name"? Answer: null
  - A string with no characters, "", is NOT null, it is just an empty string

Methods cannot be called on a null object



NullPointerException runtime error

Exercise 1: Fields and Local Variables

- Load the latest version of the barking dogs example (class Dog2)
- Identify all variables and classify them as fields or local variables
- Rename all fields to use the _ (underscore) prefix
- Explore the accessibility of the different variables in the program
  - Can you access local variables before they are declared?
  - Can you access local variables from outside their method?
  - What happens if you define an inner unnamed scope around your variable declaration?
  - Can you use a field that has not been initialized (call bark before setters)?
  - Can you use a local variable that has not been initialized?
- Note down any syntax errors you get so you know how to fix them later

Exercise 1b: null object variables

- Find a program you wrote that uses the Random class (rolling dice, flipping coins etc.)
- Locate the line that creates the Random object, e.g.

```
Random numGenerator = new Random();
```

- Assign the variable to null and run the program. What happens?
- Now change the Random object to be a field variable (not a local variable) and get your program working again
  - This is a better way to use the Random class since it generates a continuous sequence of random numbers from the same random object instead of starting a new sequence for each number
- Change your Random field so it's not initialized to anything and run the program, e.g.

```
private Random numGenerator;
```

Now, our first program with two classes...

Exercise 2: Private and Public Fields

- Continuing with Exercise 1 (class Dog2) …
- Can you access private fields outside the class they are defined in?
  - Test this by attempting to print out a private field directly from the main method in class Dog2
  - Then split the program into two classes: Dog2 and DogTest
  - Move the main method from class Dog2 into class DogTest
    - Can you still access the private field?
  - Now make the field public and see if you can access it
  - Finally, change the field back to private… then figure out how you can access it and still keep it private!

What if the value of a variable never changes?

Is this still a variable since it doesn't vary?

Constants

- Constants are variables that never change their values
  - They are variables in the sense that they "remember" information
  - The information they remember never changes
  - Example: PI = 3.14
- Used to provide a name for a value as a common identifier.
  - Avoid using hard-coded numbers or strings. Define constants instead if the values never change
- Value must be provided at the time of declaration.
- By convention constants are named using ALL_CAPITAL_LETTERS with multiple words separated by underscores
- Can be both fields and local variables. More often used as fields
- Declared with the keyword final
  - When using constant fields always use static. Constant fields are "allowed" to be public (the only exceptions)

Defining a constant in a class

Class declaration

Class definition

```
public class Circle
{
    public static final double PI = 3.14;
    …
}
```

Defines this field as a constant. Its value cannot be changed

Best Practice: The only field variables that should be public are static constant fields. All other fields should be private, can have accessor/mutator methods if needed.

Use static for 'final' fields

Variables vs. Constants

Variables

- Value can be changed after initialization
- No special keyword required to declare
- Do not need to be initialized when declared
- camelCase naming convention starting with lower case
- Fields or local variables

Constants

- Value cannot be changed after initialization
- final keyword required when declaring the constant
- Must be initialized when declared
- ALL_CAPITAL separated by _ naming convention
- Fields or local variables

Exercise 3: Constants

- Continuing with Exercise 2 (class Dog2 and DogTest)
- Change the large dog "threshold" value (30) to be a constant field instead of a hard-coded value

- This is especially useful if a constant is used many times in a program
  - Changing the constant is easy, change is needed in only one place
  - Another example: math programs may need the value Pi
    - This constant is very unlikely to ever change, it's provided for you in the Java library: Math.PI