Java Shadowing

Names

Names refer to entities declared in a program. A declared entity can be a package, class, interface, member (class, interface, field, or method) of a reference type, type parameter, formal parameter, exception parameter, or local variable.

Names are simple or qualified. Simple names are a single identifier and qualified names are sequences of identifiers separated by '.' (dot or period) tokens. Declarations within a scope can be referenced by a simple name if the declaration is not shadowed (discussed next).

```
public class Bicycle {
    int speed;
    int gear;
    int cadence;

    /// Parameters shadow class fields
    public Bicycle(int speed, int gear, int cadence) {
        this.speed = speed;
        this.gear = gear;   /// Use this keyword to refer to class field
        this.cadence = cadence;
    }

    public setCadence(int cadence) {
        this.cadence = cadence;
    }

    public void speedUp(int increment){
        int speed; /// Local variable shadows class field
        speed = (speed + increment);

        this.speed = speed;
    }
}
```

Figure 1 Bicycle class with shadowing examples.

Shadowing

Shadowing occurs where variables in one scope have the same name as another declaration in enclosing scope. The declaration in the enclosing scope cannot be referenced by a simple name. Shadowing is conventionally used only within constructors and setter methods for parameter names. To access the class field, you must use a qualified name using the keyword `this`. Outside of these cases, shadowing can lead to confusing code.

Shadowing can occur with local variables as well. In these cases, you will need to be aware which variable you are intending to use. Using **this** will refer to the field and without **this** will refer to the local variable. Shadowing with local variables should be avoided. Figure 2 shows an instance of shadowing that can occur with inner classes.

```
class Organization {
    String name;

    class SubGroup {
        String name;

        SubGroup(String name) {
            this.name = name;
        }

        void print(String name) {
            /// Parameter shadows SubGroup.name and Organization.SubGroup.name
            System.out.println(name);

            /// Use this to refer to SubGroup field
            System.out.println(this.name);

            /// Qualify this.name with Organization to refer to Organization field
            System.out.println(Organization.this.name);
        }
    }

    SubGroup sub;

    public Organization(String name, String subName) {
        this.name = name;
        sub = new SubGroup(subName);
    }
}
```

Figure 2 Shadowing of name by an inner class field

Obscuring

Some usages of a simple name can be interpreted as a variable, type, or package. When deciding which the simple name should refer to, Java does so in the listed order of precedence i.e., variable over type, and type over package. If a simple name refers to a variable, then it is impossible to refer to types or packages by the same simple name. The declarations of the types or packages are said to be obscured. Import statements can handle obscured packages.

Naming conventions help prevent obscuring.

Naming Conventions

Naming conventions provide guidance that can help prevent issues with naming from occurring such as obscuring. What follows are some conventions laid out by the Java language specification. Code provided for our labs (files and snippets) may not always follow these conventions for illustrative purposes. Some conventions are left to other labs focused on the subject (packages and generic types).

Classes and interfaces

Names of classes should be descriptive nouns or noun phrases in mixed case where the first letters of each word are capitalized. Names of interfaces can follow similar conventions or be adjectives describing behaviors e.g., `Runnable` or `Cloneable.`

Class fields and constant names

Fields which are not final should be mixed case with a lowercase first letter and the first letter of subsequent letters capitalized. Fields should be nouns, noun phrases, or abbreviations for nouns.

Constant names should be descriptive and not unnecessarily abbreviated. They should be sequences of one or more words in all uppercase with separated by underscores "_" as necessary.

Method names

Method names should be verbs or verb phrases in mixed case where the first letter is lowercase and subsequent words are capitalized.

Some specific conventions for methods include:

- Accessors and mutators of class fields for a variable `V` should be named `getV` and `setV` respectively.
- Methods which return a length should be named `length`.
- Methods which test a boolean condition `V` should be named `isV.`
- Methods which convert to a format `F` should be named `toF`.

Local variables and parameter names

These names should be short, but meaningful. They can be acronyms, abbreviations, or mnemonic terms. One character variable or parameter names should be avoided. They can be used for looping and temporary variables or when their type is clear and undistinguishable.

Graded submission Task 1

You are provided with some Java code with some naming issues that may not necessarily build. Fix the naming issues in the provided code. Where shadowing occurs, prefer to qualify the name of the shadowed variable instead of renaming either declaration. Where obscuring of a package occurs, prefer to import the package instead of renaming the obscuring variable or type or writing a fully qualified name. Where you do decide to rename, follow the naming conventions provided in the previous section. You may also introduce new variables when necessary.

** STOP **

Ensure you have saved the source files in your project. Make a copy of your Eclipse project at this point (you can do this from your system's file explorer). Name the file labelling it as Task 1. You will submit this copy at the end of this lab.

Lab wrap-up

In this lab we discussed the phenomena of shadowing and obscuring. Shadowing occurs where variables in one scope have the same name as a variable in the enclosing scope. In these cases, the variable in the outer scope cannot be referred by a simple name. They must be referred by a qualified name. Shadowed class fields are qualified using the keyword `this`.

Obscuring occurs when a simple name can be interpreted as a variable, type, or package. Java prefers to interpret them in the listed order of precedence. Thus, it can become impossible to refer to types or packages by a simple name. Obscured packages can be handled by import statements. Obscured types are rarer because of naming conventions, but if they occur, they can be handled by renaming the variable that obscured the type (usually without too much problems to the programmer).

We then presented some naming conventions that are laid out by the Java language specification.

Submission notes for graded tasks:

- Submit a zipped folder containing the Eclipse projects for each graded task.
- For each project, ensure you have included the src folder containing the .java file, the .classpath file, and the .project file.
- Ensure that each project is named appropriately to correspond with each task.
- Zip the projects together and name the zipped folder “Lab5_firstName_lastName”.
- Submit the zipped file to the respective Canvas link.

Rubrics
Task 1 (allotted points): - Compilation (2) - Shadowed names are qualified when possible (4) - Follows naming conventions (4)