

Object-Oriented Programming

Class Members

Contents

- Class methods vs. instance methods
- Class variables vs. instance variables

Class Methods

- Examples:

```
double x = Math.round(42.2);
int y = Math.abs(-10);
```

- Methods in the Math class don't use any instance variable values. So they don't need to know about a specific Math **object**. All we need is the Math **class**
- Math functions were written as **class** methods, or **static** methods
- A class method (static method) is one that runs *without any instance of the class*

Instance Methods vs. Class Methods

Instance (regular) methods

```
class Cow {  
    String name;  
    public String greeting() {  
        return ("Hi, I am " + name);  
    }  
}
```

Class (static) methods

```
class Math {  
    public static int abs(int a) {  
        if (a > 0) return a;  
        return -a;  
    }  
}
```

- Behavior of instance method *greeting()* is affected by instance variable *name*
- Instance method is called using a *reference variable*
`s = cow1.greeting();`

- Static method *abs()* cannot use instance variables of Math class
- Static method is called using the class name:
`int a = Math.abs(-10);`

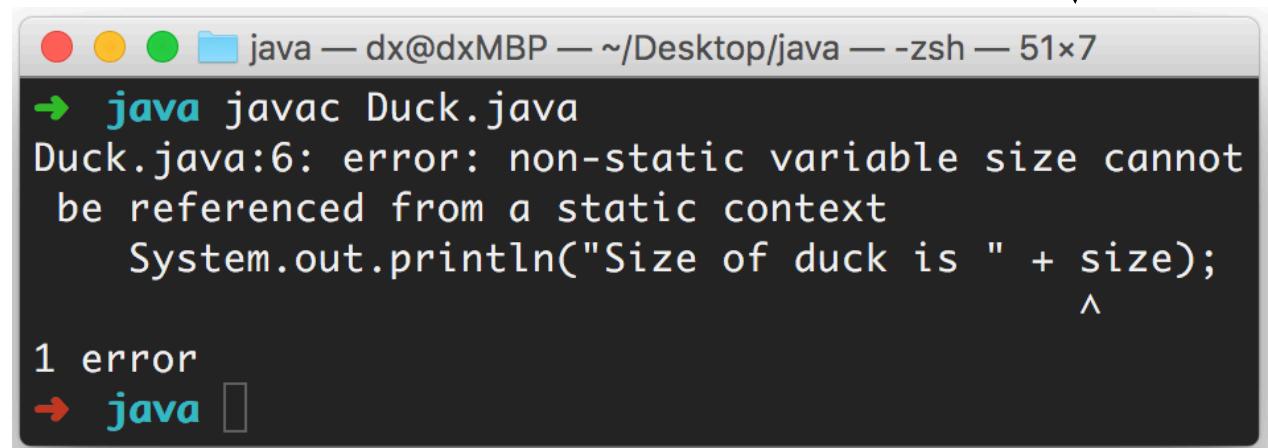
Using Class Methods

- Class (static) methods can't use:
 - instance variables
 - instance methods

Using Class Methods

```
public class Duck {  
    private int size;  
  
    public static void main( String[] args ) {  
        Duck d = new Duck();  
        System.out.println("Size of duck is " + size);  
    }  
}
```

Compile error: non-static variable “size”
can’t be referenced from a static context



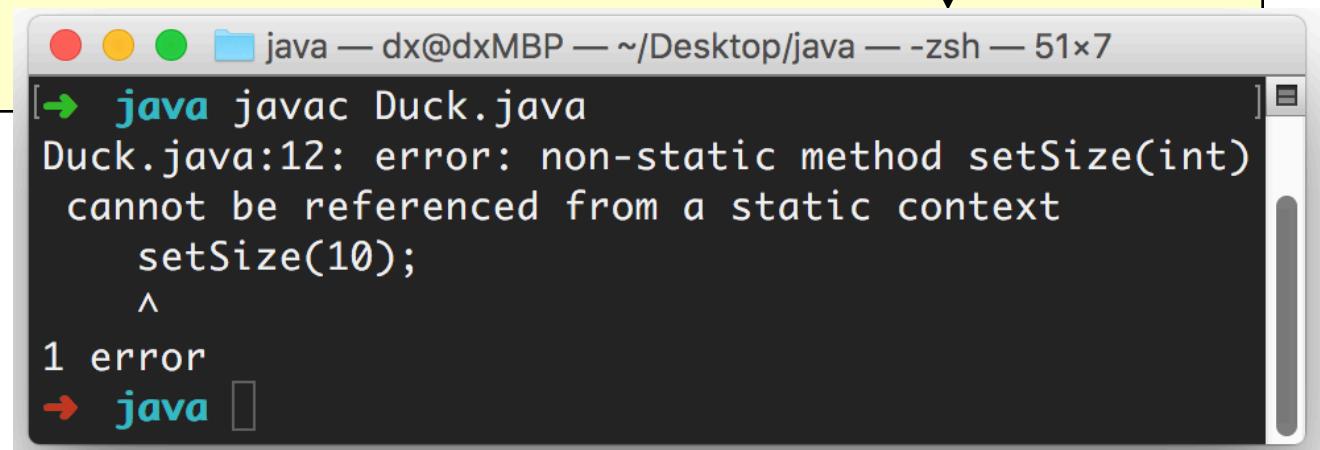
A terminal window showing the compilation of a Java file named Duck.java. The window title is "java — dx@dxMBP — ~/Desktop/java — -zsh — 51x7". The command "java javac Duck.java" is run, followed by the error message: "Duck.java:6: error: non-static variable size cannot be referenced from a static context". The error line is "System.out.println("Size of duck is " + size);". A cursor arrow points to the closing brace of the println statement. The terminal then shows "1 error" and ends with "→ java []".

```
java — dx@dxMBP — ~/Desktop/java — -zsh — 51x7  
→ java javac Duck.java  
Duck.java:6: error: non-static variable size cannot  
be referenced from a static context  
    System.out.println("Size of duck is " + size);  
                                         ^  
1 error  
→ java []
```

Using Class Methods

```
public class Duck {  
  
    private int size;  
  
    public static void main( String[] args) {  
        Duck d = new Duck();  
        setSize(10);  
    }  
    public void setSize (int s) {  
        if (s>0) size = s;  
    }  
}
```

Compile error: non-static method “**setSize()**”
can’t be referenced from a static context



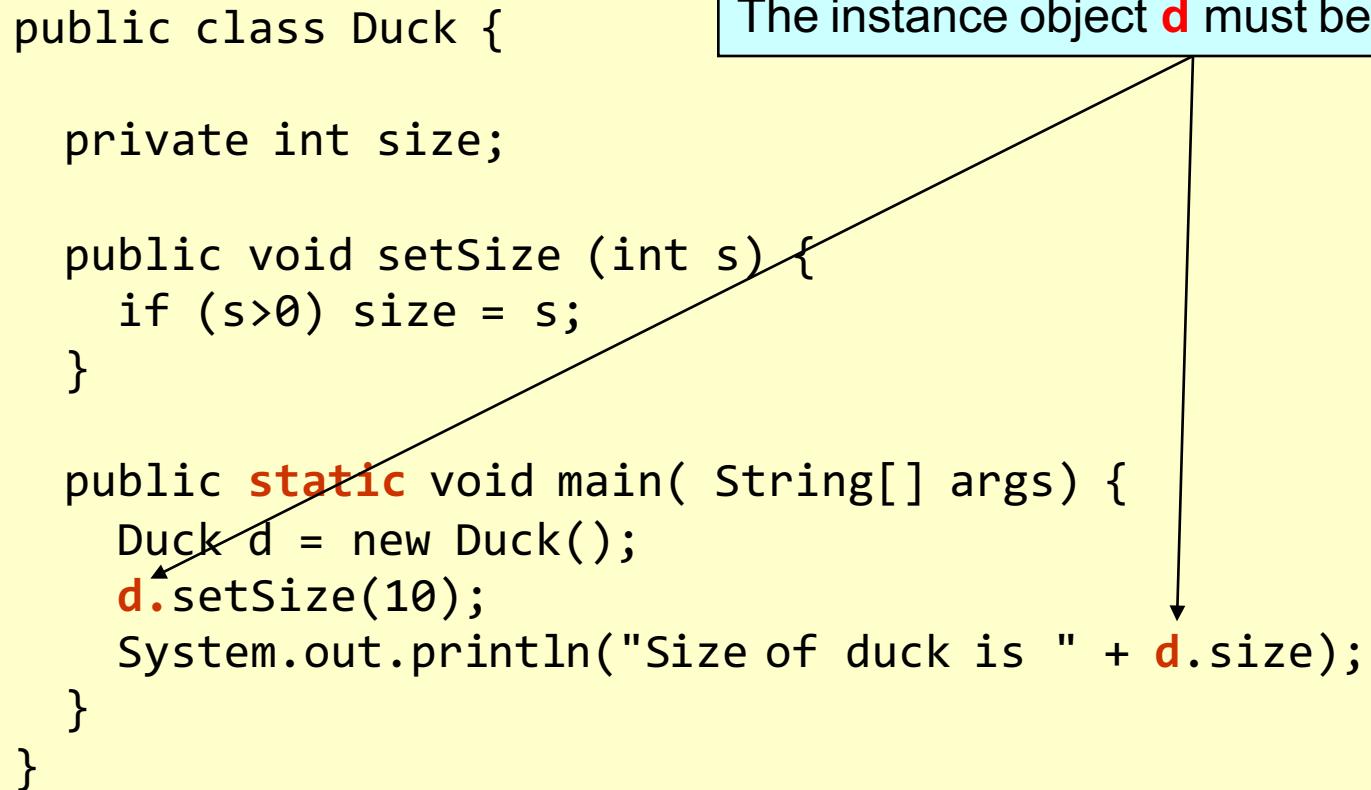
A terminal window showing the output of a Java compilation command. The window title is "java — dx@dxMBP — ~/Desktop/java — -zsh — 51x7". The command entered is "java javac Duck.java". The output shows an error message: "Duck.java:12: error: non-static method setSize(int) cannot be referenced from a static context setSize(10); ^ 1 error". The terminal prompt is "→ java □".

```
[→ java javac Duck.java  
Duck.java:12: error: non-static method setSize(int)  
cannot be referenced from a static context  
    setSize(10);  
    ^  
1 error  
→ java □
```

Correct Code

```
public class Duck {  
  
    private int size;  
  
    public void setSize (int s) {  
        if (s>0) size = s;  
    }  
  
    public static void main( String[] args) {  
        Duck d = new Duck();  
        d.setSize(10);  
        System.out.println("Size of duck is " + d.size);  
    }  
}
```

The instance object **d** must be specified



The diagram shows a light blue box containing the error message "The instance object **d** must be specified". Two arrows point from this box to the variable **d** in the code: one from the word "instance" to the **d** in "d.setSize(10)", and another from the word "object" to the **d** in "d.size".

Better Code

- The program is put in a separate class:
 - Class Duck should define Duck objects only
 - Different programs can use the same Duck class

```
public class Duck {  
    private int size;  
    public void setSize (int s) {...}  
    ...  
    public class DuckProgram {  
        public static void main( String[] args) {  
            Duck d = new Duck();  
            d.setSize(10);  
            System.out.println("Size of duck is " + d.size);  
        }  
    }  
}
```

Class Variables

- A class variable (or static variable) belongs to the class, not any object
- Need just one copy, but shared among all class instances

```
public class Duck {  
  
    private int size;  
    public static int count = 0;  
  
    public Duck() {  
        count++;  
    }  
    ...  
}
```

Each duck has its own size.
But all ducks share the same attribute “count”

Class Variables vs. Instance Variables

Class/static variables

- belong to a class
- need just one copy, but shared among all instances of the class
- initialized before any objects of the class

Instance variables

- belong to an instance
- each instance has its own copy
- initialized when the owner object is created

```
public class Duck {  
    private int size = 0;  
    public static int count = 0;  
  
    public Duck() {  
        count++;  
        size++;  
    }  
}
```

Using Class Variables

- Class (static) variables can be used by:
 - static methods
 - instance methods

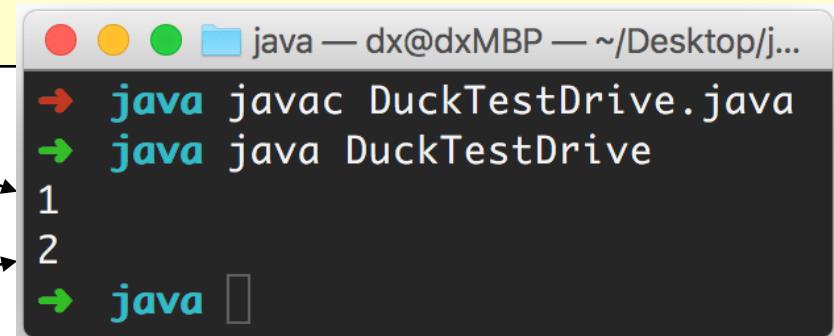
Using Class Variables

```
public class Duck {  
  
    private int size;  
    public static int count = 0;  
  
    public void incCount()  
    {  
        count++;  
    }  
  
}
```

```
public class DuckTestDrive {  
    public static void main(String [] args) {  
        Duck d = new Duck();  
        d.incCount();  
        System.out.println(d.count);  
        d.incCount();  
        System.out.println(Duck.count);  
    }  
}
```

Static variable **count** is called by instance object **d**

Static variable **count** is called by class name **Duck**



A terminal window showing the execution of Java code. The window has a dark background with light-colored text. It shows the command `javac DuckTestDrive.java` followed by the output of the program, which is `1` on the first line and `2` on the second line. The terminal window has a title bar with the text "java — dx@dxMBP — ~/Desktop/j..." and a close button.

```
java — dx@dxMBP — ~/Desktop/j...  
→ java javac DuckTestDrive.java  
→ java java DuckTestDrive  
1  
2  
→ java
```

