Creating Objects

- A variable holds either a primitive value or a *reference* to an object

- A class name can be used as a type to declare an *object reference variable*

```
String title;
```

- No object is created with this declaration

- An object reference variable holds the address of an object

- The object itself must be created separately

Creating Objects

- Generally, we use the `new` operator to create an object

- Creating an object is called *instantiation*

- An object is an *instance* of a particular class

```
title = new String("Java Software Solutions");
```

**This calls the String *constructor*, which is
a special method that sets up the object**

Invoking Methods

- We've seen that once an object has been instantiated, we can use the *dot operator* to invoke its methods

```
numChars = title.length()
```

- A method may *return a value*, which can be used in an assignment or expression

- A method invocation can be thought of as asking an object to perform a service

References

- Note that a primitive variable contains the value itself, but an object variable contains the address of the object

- An object reference can be thought of as a pointer to the location of the object

- Rather than dealing with arbitrary addresses, we often depict a reference graphically

num1   `3 8`

name1   → `"Steve Jobs"`
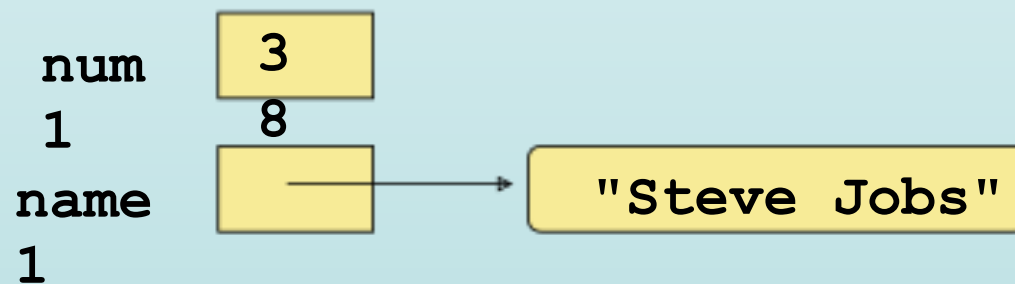
Assignment Revisited

- The act of assignment takes a copy of a value and stores it in a variable

- For primitive types:

Before: num1 `3 8` num2 `9 6`
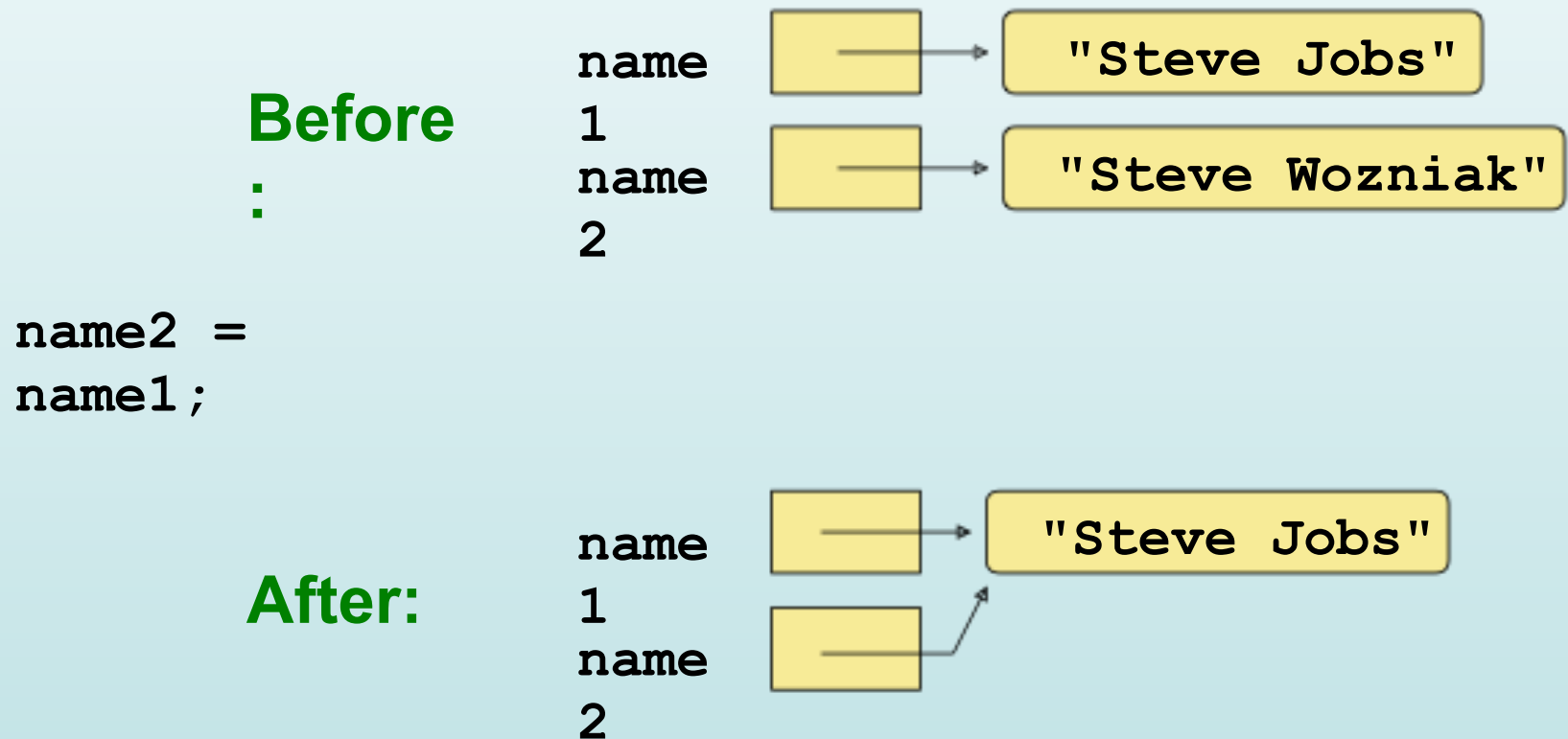
```
num2 = num1;
```

After: num1 `3 8` num2 `3 8`

Reference Assignment

- For object references, assignment copies the address:

Before:

```
name1  [  ] ─────────→ "Steve Jobs"
name2  [  ] ─────────→ "Steve Wozniak"
```

```
name2 = name1;
```

After:

```
name1  [  ] ─────────→ "Steve Jobs"
name2  [  ] ─────────↗
```

Aliases

- Two or more references that refer to the same object are called *aliases* of each other

- That creates an interesting situation: one object can be accessed using multiple reference variables

- Aliases can be useful, but should be managed carefully

- Changing an object through one reference changes it for all of its aliases, because there is really only one object

Garbage Collection

- When an object no longer has any valid references to it, it can no longer be accessed by the program

- The object is useless, and therefore is called *garbage*

- Java performs *automatic garbage collection* periodically, returning an object's memory to the system for future use

- In other languages, the programmer is responsible for performing garbage collection

The String Class

- Because strings are so common, we don't have to use the `new` operator to create a `String` object

  ```
  title = "Java Software Solutions";
  ```

- This is special syntax that works <u>only</u> for strings

- Each string literal (enclosed in double quotes) represents a `String` object

String Methods

- Once a `String` object has been created, neither its value nor its length can be changed

- Therefore we say that an object of the `String` class is *immutable*

- However, several methods of the `String` class return new `String` objects that are modified versions of the original

Class Libraries

- A *class library* is a collection of classes that we can use when developing programs

- The *Java standard class library* is part of any Java development environment

- Its classes are not part of the Java language per se, but we rely on them heavily

- Various classes we've already used (`System`, `Scanner`, `String`) are part of the Java standard class library

Packages

- For purposes of accessing them, classes in the Java API are organized into *packages*

- These often overlap with specific APIs

- Examples:

Package	Purpose
java.lang	General support
java.util	Utilities
java.net	Network communication
javafx.scene.shape	Graphical shapes
javafx.scene.control	GUI controls

The import Declaration

- When you want to use a class from a package, you could use its *fully qualified name*

```
java.util.Scanner
```

- Or you can *import* the class, and then use just the class name

```
import java.util.Scanner;
```

- To import all classes in a particular package, you can use the * wildcard character

```
import java.util.*;
```

The import Declaration

- All classes of the `java.lang` package are imported automatically into all programs

- It's as if all programs contain the following line:

$$\text{import java.lang.*;}$$

- That's why we didn't have to import the `System` or `String` classes explicitly in earlier programs

- The `Scanner` class, on the other hand, is part of the `java.util` package, and therefore must be imported