

Lecture 3: Using Classes and Objects

CSC 1214: Object-Oriented Programming

Using Classes and Objects

- We can create OO programs using predefined classes and related objects
- Outline
 - An overview of the Java programming language
 - object creation and object references
 - the String class and its methods
 - the Java standard class library
 - the Random and Math classes

First, an Overview of the Java Programming Language

The Java Programming Language

Lecture 3

CSC 1214 - Object-oriented Programming

Identifiers

- ❖ *Identifiers* are the words a programmer uses in a program
- ❖ An identifier can be made up of letters, digits, the underscore character (`_`), and the dollar sign (`$`)
- ❖ Identifiers cannot begin with a digit
- ❖ Java is *case sensitive* - `Total`, `total`, and `TOTAL` are different identifiers
- ❖ By convention, programmers use different case styles for different types of identifiers, such as
 - *title case* for class names - `Welcome`
 - *upper case* for constants - `MAXIMUM`

Identifiers

- ❖ Sometimes we choose identifiers ourselves when writing a program (such as `Welcome`)
- ❖ Sometimes we are using another programmer's code, so we use the identifiers that he or she chose (such as `println`)
- ❖ Often we use special identifiers called *reserved words* that already have a predefined meaning in the language
- ❖ A reserved word cannot be used in any other way

Java Reserved words

<code>abstract</code>	<code>else</code>	<code>int</code>	<code>strictfp</code>
<code>boolean</code>	<code>enum</code>	<code>interface</code>	<code>super</code>
<code>break</code>	<code>extends</code>	<code>long</code>	<code>switch</code>
<code>byte</code>	<code>false</code>	<code>native</code>	<code>synchronized</code>
<code>case</code>	<code>final</code>	<code>new</code>	<code>this</code>
<code>catch</code>	<code>finally</code>	<code>null</code>	<code>throw</code>
<code>char</code>	<code>float</code>	<code>package</code>	<code>throws</code>
<code>class</code>	<code>for</code>	<code>private</code>	<code>transient</code>
<code>const</code>	<code>goto</code>	<code>protected</code>	<code>true</code>
<code>continue</code>	<code>if</code>	<code>public</code>	<code>try</code>
<code>default</code>	<code>implements</code>	<code>return</code>	<code>void</code>
<code>do</code>	<code>import</code>	<code>short</code>	<code>volatile</code>
<code>double</code>	<code>instanceof</code>	<code>static</code>	<code>while</code>

Errors

- ❖ You make mistakes when programming
- ❖ A program can have three types of errors
 - *Compile-time errors*: the compiler will find syntax errors and other basic problems
 - If compile-time errors exist, an executable version of the program is not created
 - *Run-time errors*: a problem can occur during program execution, such as trying to divide by zero, which causes a program to terminate abnormally
 - *Logical errors* : a program may run, but produce incorrect results, perhaps using an incorrect formula
- ❖ Errors in programs are often called *bugs*

Keep your cool



- ❖ You will have errors
- ❖ You will correct them
- ❖ Seek help if you don't understand the error.

Java Data types

❖ all data is either

➤ an object *i.e.* an instance of some class

➤ a primitive data type

- int
- float, double
- char
- boolean

Java Data types

❖ Java is *strongly, statically typed*

➤ strongly typed: all data has a type

➤ statically typed: all types must be declared before use

❖ type declarations can occur anywhere in source code

```
int foo;    // foo has type int
```

Java Data types

- ❖ `int` → integers
- ❖ `float` → single precision floating point
- ❖ `double` → double precision floating point
- ❖ `char` → Unicode characters (16 bit)
- ❖ `boolean` → `true` or `false` (not 0 or 1)
- ❖ `byte` → 8 bits; "raw data"
- ❖ `String` → character strings

Operators

❖ like in C:

➤ $+ \ - \ * \ / \ \% \ = \ ++ \ -- \ += \ -=$ etc.

❖ precedence:

➤ Parenthesis

➤ Multiplication and division

➤ Addition and subtraction

- $a + b * c \rightarrow a + (b * c)$ NOT $(a + b) * c$
- use parentheses if need to override defaults

Control Flow Statements

- Decision-making statements / conditionals
 - if, if-else, switch
- The looping statements
 - for, while, do – while
- The branching statements
 - break, continue, return

Similar to C, we will discuss these in the next lectures

Back to, Using Classes and Objects



Outline



Creating Objects

The String Class

Packages and Class Libraries

Creating Objects

- A variable holds either a primitive type 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**

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



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

- **Creating an object is called *instantiation***
- **An object is an *instance* of a particular class**

Invoking Methods

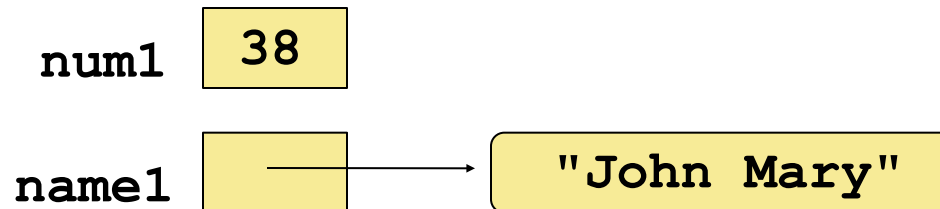
- Once an object has been instantiated, we can use the *dot operator* to invoke its methods

```
int count = 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**



Assignment Revisited

- The act of assignment takes a copy of a value and stores it in a variable
- For primitive types:

Before:

num1	38
num2	96

`num2 = num1 ;`

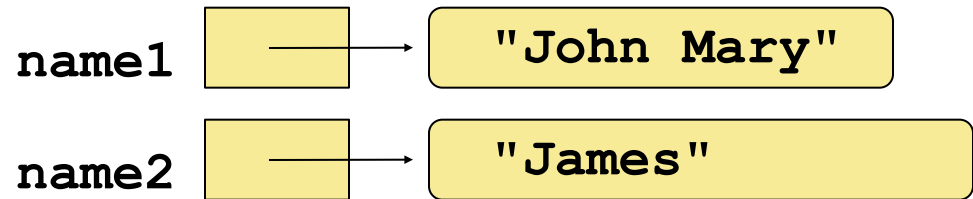
After:

num1	38
num2	38

Reference Assignment

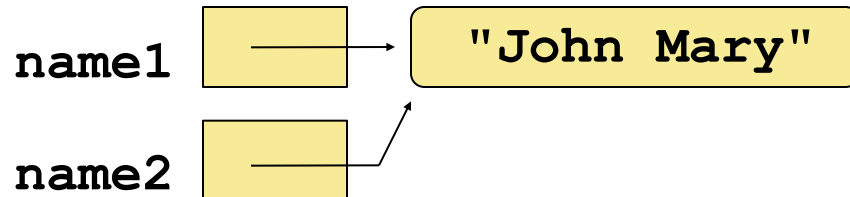
- For object references, assignment copies the address:

Before:



`name2 = name1;`

After:



Aliases

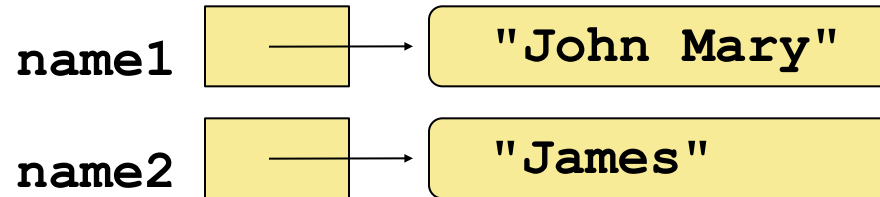
- **Two or more references that refer to the same object are called *aliases* of each other**
- **That creates an interesting situation: one object can be accessed using multiple reference variables**
- **Aliases can be useful, but should be managed carefully**
- **Changing an object through one reference 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**

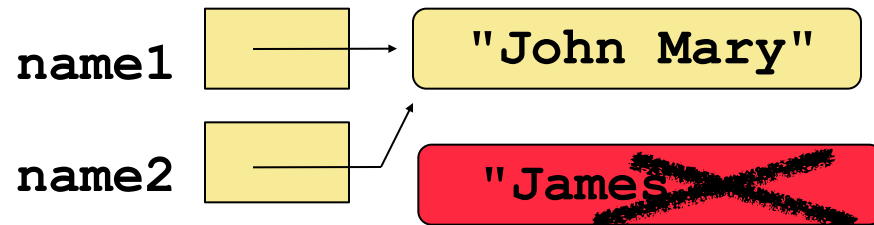
Garbage Collection

Before:



`name2 = name1;`

After:



Garbage

Outline

Creating Objects



The String Class

Packages and Class Libraries

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 only 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
- Thus 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
- See the list of **String** methods on page 89 of the course textbook

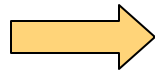
String Indexes

- It is occasionally helpful to refer to a particular character within a string
- This can be done by specifying the character's numeric *index*
- The indexes begin at zero in each string
- In the string "Hello", the character 'H' is at index 0 and the 'o' is at index 4
- See StringMutation.java (page 90) of the course textbook

Outline

Creating Objects

The String Class



Packages and Class Libraries

Class Libraries

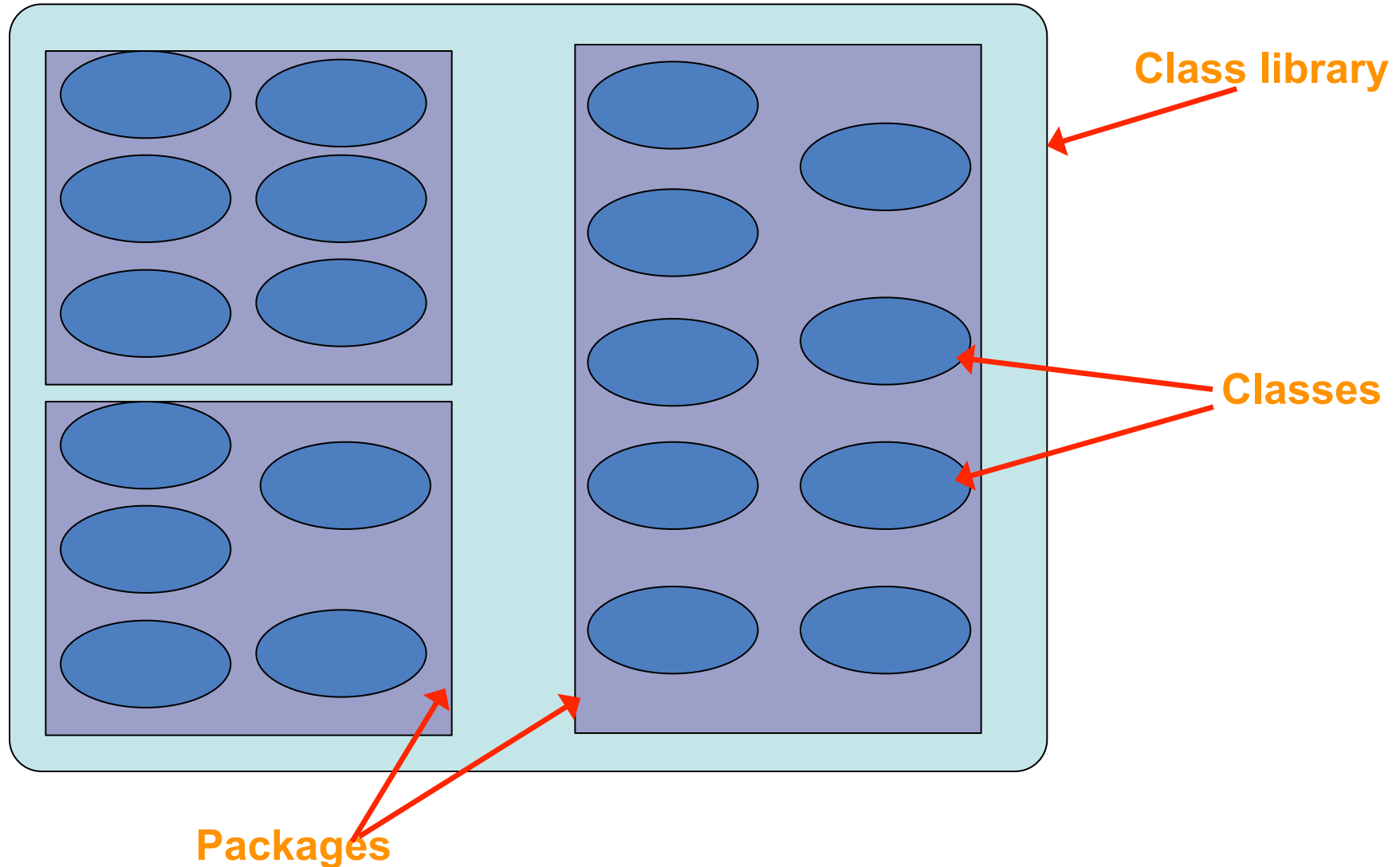
- A class library is a collection of classes that we can use when developing programs
- The Java standard class library is part of any Java development environment
- Its classes are not part of the Java language per se, but we rely on them heavily
- The **System** class and the **String** class (that we have already used) are part of the Java standard class library
- Other class libraries can be obtained through third party vendors, or you can create them yourself

Packages

- The classes of the Java standard class library are organized into packages
- Some of the packages in the standard class library are:

<u>Package</u>	<u>Purpose</u>
java.lang	General support
java.applet	Creating applets for the web
java.awt	Graphics and graphical user interfaces
javax.swing	Additional graphics capabilities and components
java.net	Network communication
java.util	Utilities
javax.xml.parsers	XML document processing

Class Libraries & Packages



Working With Packages

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

```
java.util.Random
```

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

```
import java.util.Random;
```

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

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

Working With Packages

- All classes of the `java.lang` package are imported automatically into all programs
- That's why we didn't have to import the `System` or `String` classes explicitly in earlier programs
- Other classes must be imported or referred to using its fully qualified name.

Example: The Random Class

- The Random class is part of the `java.util` package
- It provides methods that generate pseudorandom numbers
- A Random object performs complicated calculations based on a *seed value* to produce a stream of seemingly random values
- See RandomNumbers.java (page 97)

Example: The Math Class

- **The `Math` class is part of the `java.lang` package**
- **The `Math` class contains methods that perform various mathematical functions**
- **These include:**
 - **absolute value**
 - **square root**
 - **exponentiation**
 - **trigonometric functions**

Example: The Math Class

- The methods of the `Math` class are *static methods* (also called *class methods*)
- Static methods can be invoked through the class name – no object of the `Math` class is needed

```
value = Math.cos(90) + Math.sqrt(delta);
```

- See `Quadratic.java` (page 102) of the course textbook
- We will discuss static methods further in the next lectures

Exercises

- **Write an application that reads a person's age as an integer and prints their year of birth.**
- **Write an application that reads an integer value and prints its square root.**