CSE 1201
Object Oriented Programming

Using Classes and Objects

Acknowledgement

- **Course materials from Dr. Tagrul Dayar, Bilkent University**

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

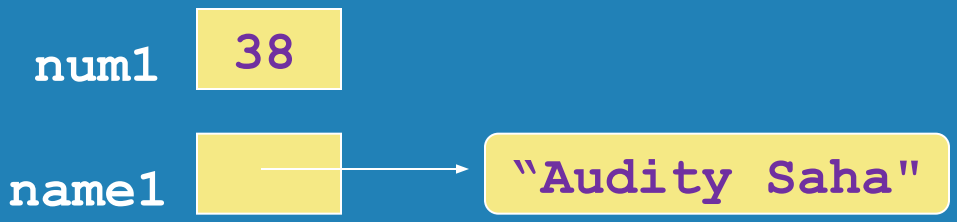
- We've seen that once an object has been instantiated, we can use the *dot operator* to invoke its methods

```
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

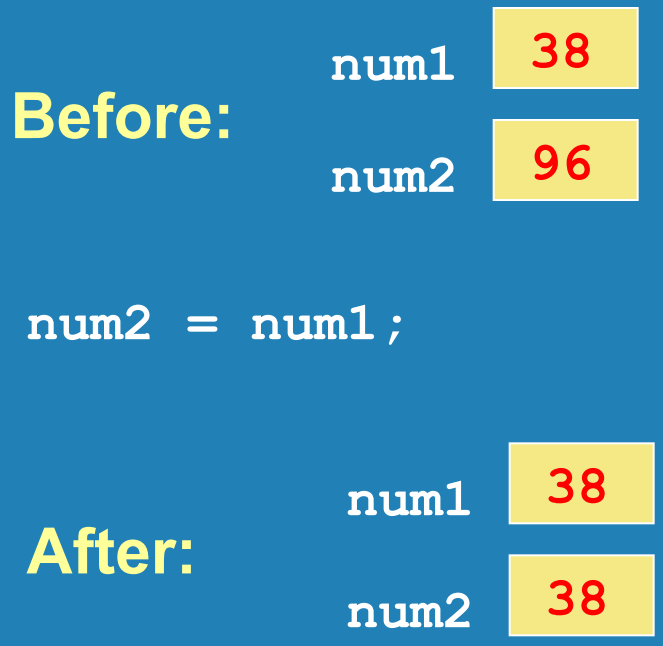
References

- 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

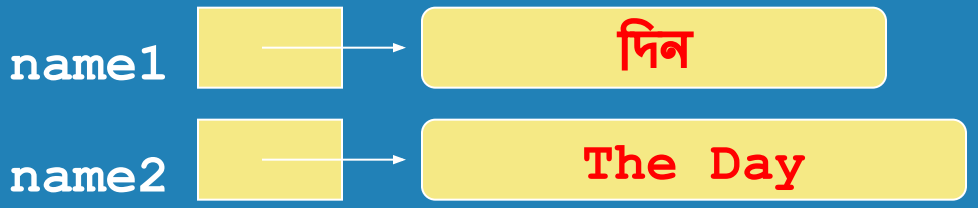
- The act of assignment takes a copy of a value and stores it in a variable
- For primitive types:



Reference Assignment

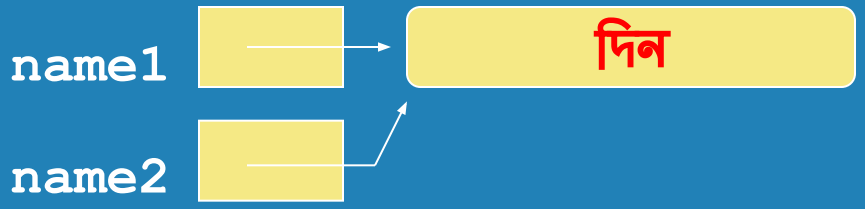
- For object references, assignment copies the address:

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

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

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

Example

```
String phrase = new String ("Change is inevitable");
String mutation1, mutation2, mutation3, mutation4;

System.out.println ("Original string: \"" + phrase + "\"");
System.out.println ("Length of string: " + phrase.length());

mutation1 = phrase.concat (", except from vending machines.");
mutation2 = mutation1.toUpperCase();
mutation3 = mutation2.replace ('E', 'X');
mutation4 = mutation3.substring (3, 30);

// Print each mutated string
System.out.println ("Mutation #1: " + mutation1);
System.out.println ("Mutation #2: " + mutation2);
System.out.println ("Mutation #3: " + mutation3);
System.out.println ("Mutation #4: " + mutation4);

System.out.println ("Mutated length: " + mutation4.length());
```

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
- 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:

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

The import Declaration

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

```
java.util.Scanner;

java.util.Random;
```

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:

```
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

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

```
  Random generator = new Random();
int num1;
float num2;

num1 = generator.nextInt();
System.out.println ("A random integer: " + num1);

num1 = generator.nextInt(10);
System.out.println ("From 0 to 9: " + num1);

num1 = generator.nextInt(15) + 20;
System.out.println ("From 20 to 34: " + num1);

num1 = generator.nextInt(20) - 10;
System.out.println ("From -10 to 9: " + num1);

num2 = generator.nextFloat();
System.out.println ("A random float [between 0-1]: " + num2);

num2 = generator.nextFloat() * 6;  // 0.0 to 5.999999
num1 = (int) num2 + 1;
System.out.println ("From 1 to 6: " + num1);
```

Interactive Programs

- Programs generally need input on which to operate
- The `Scanner` class provides convenient methods for reading input values of various types
- A `Scanner` object can be set up to read input from various sources, including the user typing values on the keyboard
- Keyboard input is represented by the `System.in` object

Reading Input

- The following line creates a Scanner object that reads from the keyboard:

```
Scanner scan = new Scanner (System.in);
```

- The `new` operator creates the `Scanner` object
- Once created, the `Scanner` object can be used to invoke various input methods, such as:

```
answer = scan.nextLine();
```

Reading Input

- The `Scanner` class is part of the `java.util` class library, and must be imported into a program to be used
- See Echo.java (page 91)
- The `nextLine` method reads all of the input until the end of the line is found

Echo.java

```
import java.util.Scanner;

public class Echo {
    //  Reads a character string from the user and prints it.
    public static void main (String[] args) {
     String message;
     Scanner scan = new Scanner (System.in);

     System.out.println ("Enter a line of text:");
     message = scan.nextLine();

     System.out.println ("You entered: \"" + message + "\"");
   }
}
```

Input Tokens

- Unless specified otherwise, *white space* is used to separate the elements (called *tokens*) of the input
- White space includes space characters, tabs, new line characters
- The `next` method of the `Scanner` class reads the next input token and returns it as a string
- Methods such as `nextInt` and `nextDouble` read data of particular types

```
public static void main (String[] args)   {
    int miles;
    double gallons, mpg;

    Scanner scan = new Scanner (System.in);

    System.out.print ("Enter the number of miles: ");
    miles = scan.nextInt();

    System.out.print ("Enter the gallons of fuel used: ");
    gallons = scan.nextDouble();

    mpg = miles / gallons;

    System.out.println ("Miles Per Gallon: " + mpg);
  }
```

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

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

```
int a, b, c;  // ax^2 + bx + c
double discriminant, root1, root2;

Scanner scan = new Scanner (System.in);

System.out.print ("Enter the coefficient of x squared: ");
a = scan.nextInt();

System.out.print ("Enter the coefficient of x: ");
b = scan.nextInt();

System.out.print ("Enter the constant: ");
c = scan.nextInt();

discriminant = Math.pow(b, 2) - (4 * a * c);
root1 = ((-1 * b) + Math.sqrt(discriminant)) / (2 * a);
root2 = ((-1 * b) - Math.sqrt(discriminant)) / (2 * a);
```