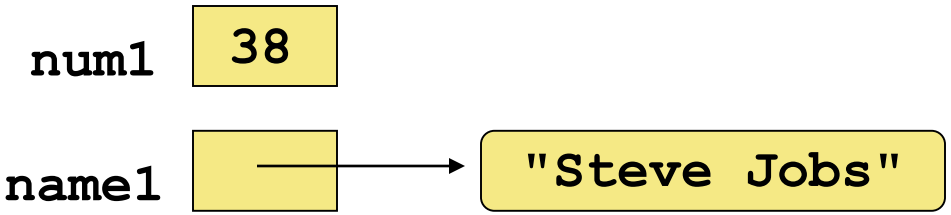
Objects

- Declaration:
- String title;
  - title (object variable) of type String(Class)
  - title is just reference (holds the address)
  - No object is created with this declaration
- Creation/Instantiation:
- title = new String (“Cin Ali");
  - title (object) is an instance of String (class)
  - BTW, only for strings, title=“Cin Ali” was enough
- Call Method : Dot operator
- count = title.length();

References

- 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



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;
```

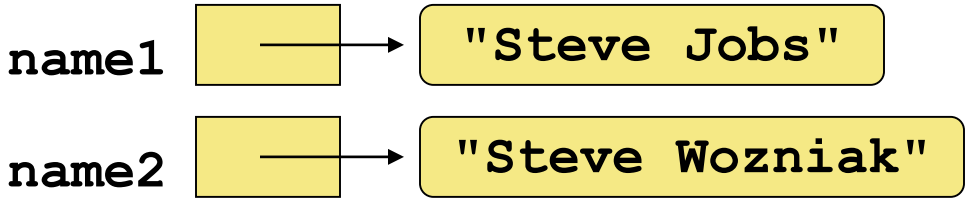
After:

`num1`	38
`num2`	38

Reference Assignment

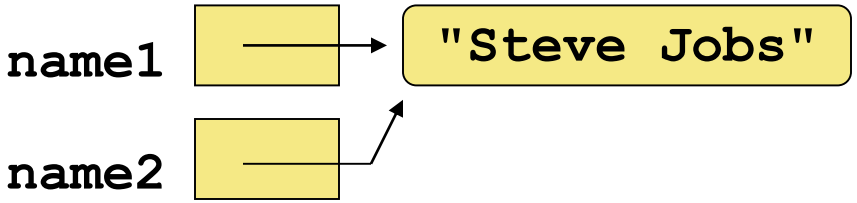
- For object references, assignment copies the address:

Before:




```
name2 = name1;
```

After:



- Changing an object through one reference changes it for all

Garbage Collection

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

Class Libraries

- Predefined classes - libraries
- Java standard class library is part of any Java development environment
  - Various classes we've already used (System, Scanner, String) are part of the Java standard class library
- It has packages

Package	Purpose
java.lang	General support (automatically imported)
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

- To use a class from a package, you could use its fully qualified name
- java.util.Scanner
- Or import the class, and then use just the class name
- import java.util.Scanner;
- To import all classes in a particular package, use the * wildcard character
- import java.util.*;

The import Declaration

- java.lang package (e.g., System, String classes) are imported automatically into all programs.
  - No need to say: import java.lang.*;
- The Scanner class is in java.util, and therefore must be imported
- Random class in java.util
- Math class in java.lang

Classes and Objects

- Die class - the blueprint for a die object
  - We can then instantiate many die objects
- State: which face is showing
  - private int faceValue
- A behavior: it can be rolled
  - roll method that assigns a random value to faceValue
- Die constructor sets faceValue of each new die object to 1
- Other methods (behaviors) that might be useful

Data Scope

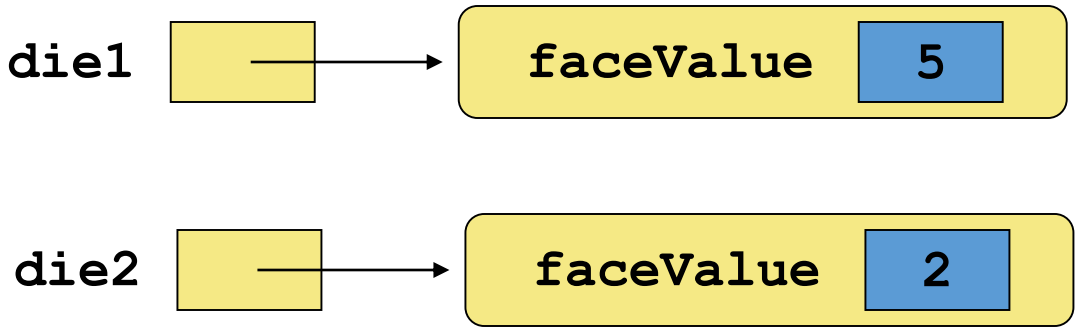
- The scope of data is the area in a program in which that data can be referenced (used)
- Data declared at the class level can be referenced by all methods in that class
- Data declared within a method (local data) can be used only in that method
- faceValue is instance data
  - each instance (object) has its own version

Instance Data

- A class declares the type of the data, but it does not reserve any memory space for it
- Every time a Die object is created, a new faceValue variable is created as well
- The objects of a class share the method definitions, but each object has its own data space
- That's the only way two objects can have different states

Instance Data

- We can depict the two Die objects from the RollingDice program as follows:



Each object maintains its own `faceValue` variable, and thus its own state

Encapsulation

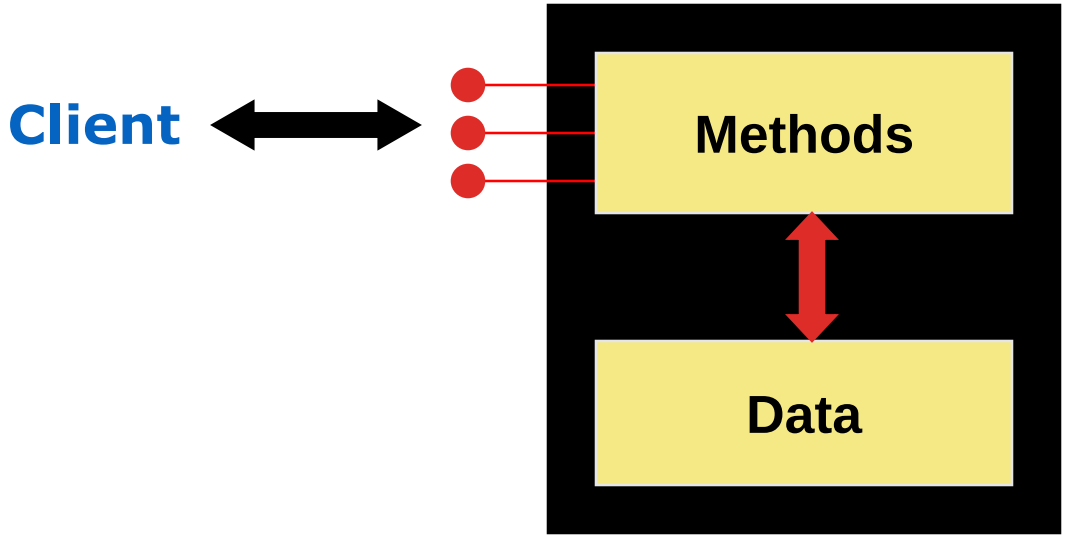
- We can take one of two views of an object:
  - internal - the details of the variables and methods of the class that defines it
  - external - the services that an object provides and how the object interacts with the rest of the system
- From the external view, an object is an encapsulated entity, providing a set of specific services
- These services define the interface to the object

Encapsulation

- One object (called the client) may use another object for the services it provides
- The client of an object may request its services (call its methods), but it should not have to be aware of how those services are accomplished
- Any changes to the object's state (its variables) should be made by that object's methods
- We should make it difficult, if not impossible, for a client to access an object’s variables directly
- That is, an object should be self-governing

Encapsulation

- An encapsulated object can be thought of as a black box -- its inner workings are hidden from the client
- The client invokes the interface methods of the object, which manages the instance data



Visibility Modifiers for Encapsulation

- public
- protected
- private
  - can be referenced only within that class
- public variables violate encapsulation
  - clients modify the values directly!!
  - Instance variables should not be declared public
- Service methods are public (for clients)
- Support methods are not public (for service methods)

Accessors and Mutators

- Because instance data is private, a class usually provides services to access and modify data values
- Accessor method returns the current value of a variable (getX, where X is the name of the value)
- Mutator method changes the value of a variable (setX)

Method Body

- The method header is followed by the method body

```
char calc (int num1, int num2, String message)
{
    int sum = num1 + num2;
    char result = message.charAt (sum);

    return result;
}
```

The return expression must be consistent with the return type

`sum` **and** `result` **are local data**

They are created each time the method is called, and are destroyed when it finishes executing

Parameters

- When a method is called, the actual parameters in the invocation are copied into the formal parameters in the method header

```
ch = obj.calc (3, count, "Hello");
```

```
char calc (int num1, int num2, String message)
{
    int sum = num1 + num2;
    char result = message.charAt (sum);

    return result;
}
```

Local Data

- Local variables inside a method are destroyed when it finishes
- Instance variables, declared at the class level, exists as long as the object exists

Constructors Revisited

- Note that a constructor has no return type specified in the method header, not even void
- A common error is to put a return type on a constructor, which makes it a “regular” method that happens to have the same name as the class
- The programmer does not have to define a constructor for a class
- Each class has a default constructor that accepts no parameters

Bank Account Example

- Let’s look at another example that demonstrates the implementation details of classes and methods
- We’ll represent a bank account by a class named Account
- It’s state can include the account number, the current balance, and the name of the owner
- An account’s behaviors (or services) include deposits and withdrawals, and adding interest

Bank Account Example