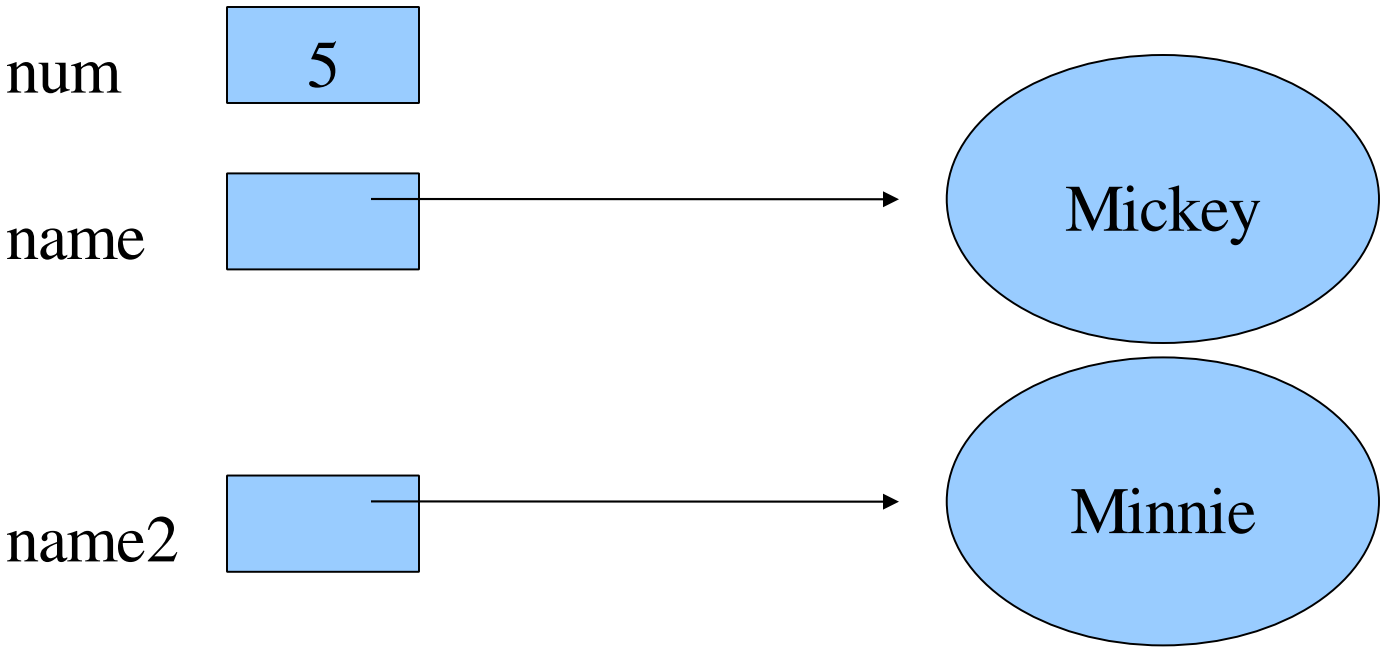
Classes/Objects

Creating Objects

- Much like creating variables of primitive types
  - String name;
  - *type name;*
- Object variables hold *references*, not values
  - can be initialized to *null*
- Use *new* to *instantiate* new objects/initialize object variables
  - String name = new String(“Mickey”);
  - invokes *Constructor* of String class

Example

```
int num = 5;
String name = new String("Mickey");
String name2 = "Minnie"; //String shortcut
```



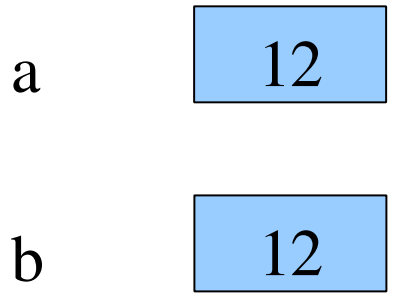
Dot Operator

- Use dot operator to access methods of an object
  - name.length()

Aliases

- Two object variables can point to the same object
- With primitive types, a copy is made

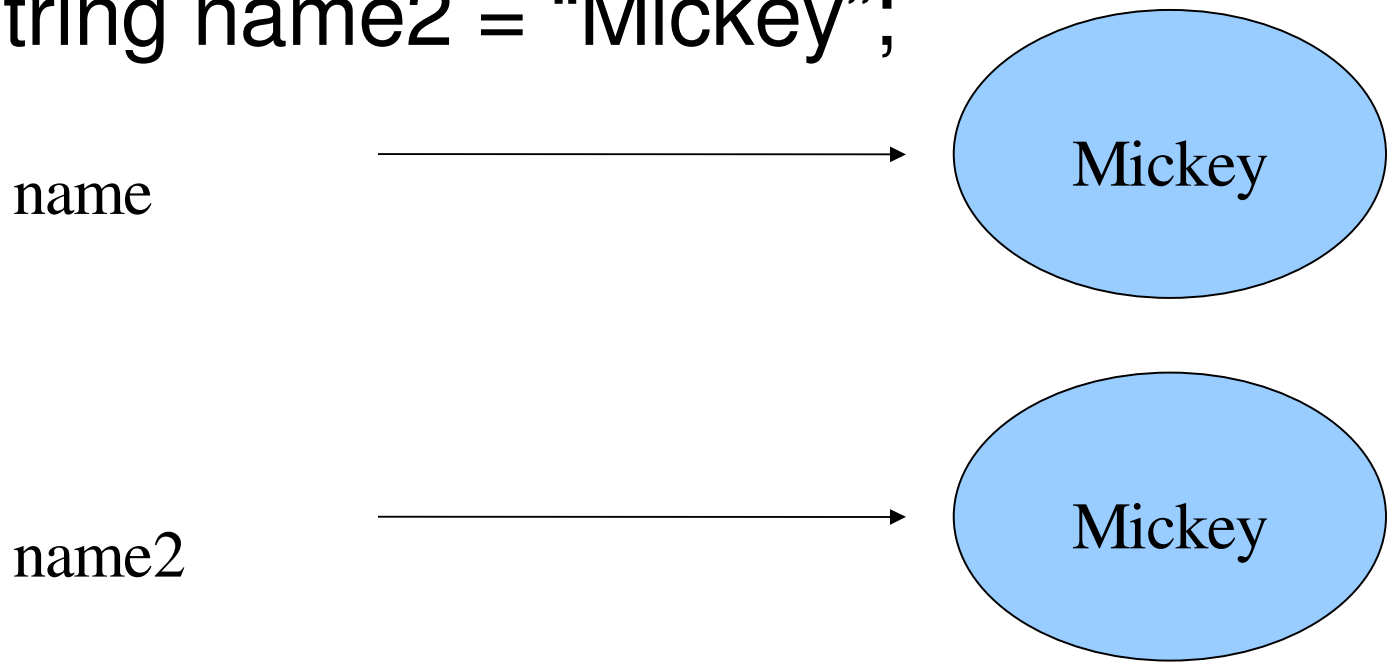
```
int a = 5;
int b = 12;
a = b;
```



Aliases

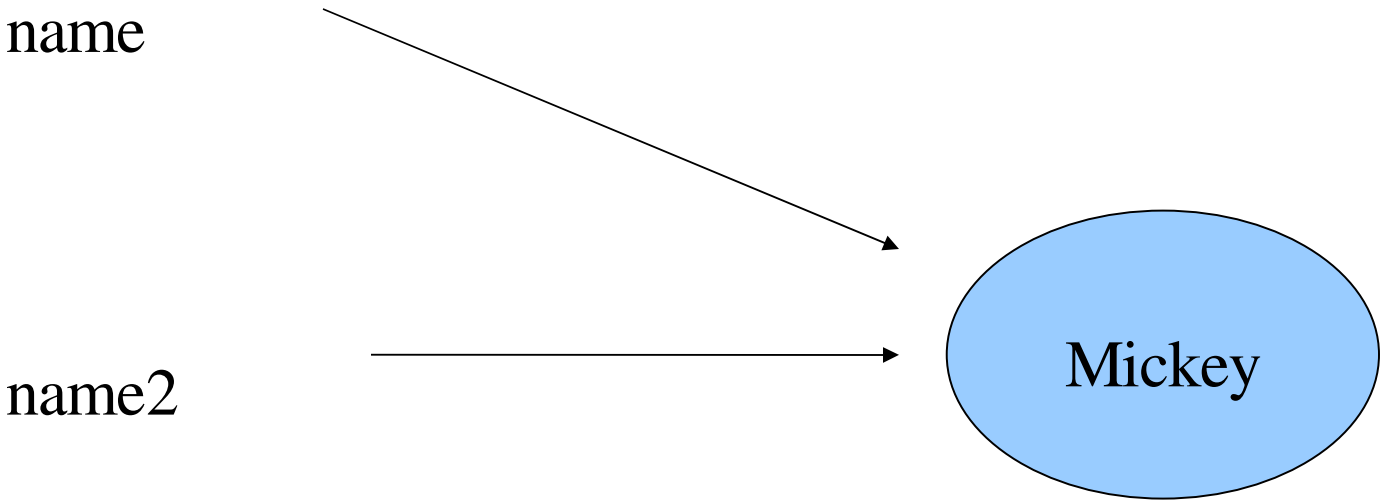
String name = “Mickey”;

String name2 = “Mickey”;



Aliases

name = name2;



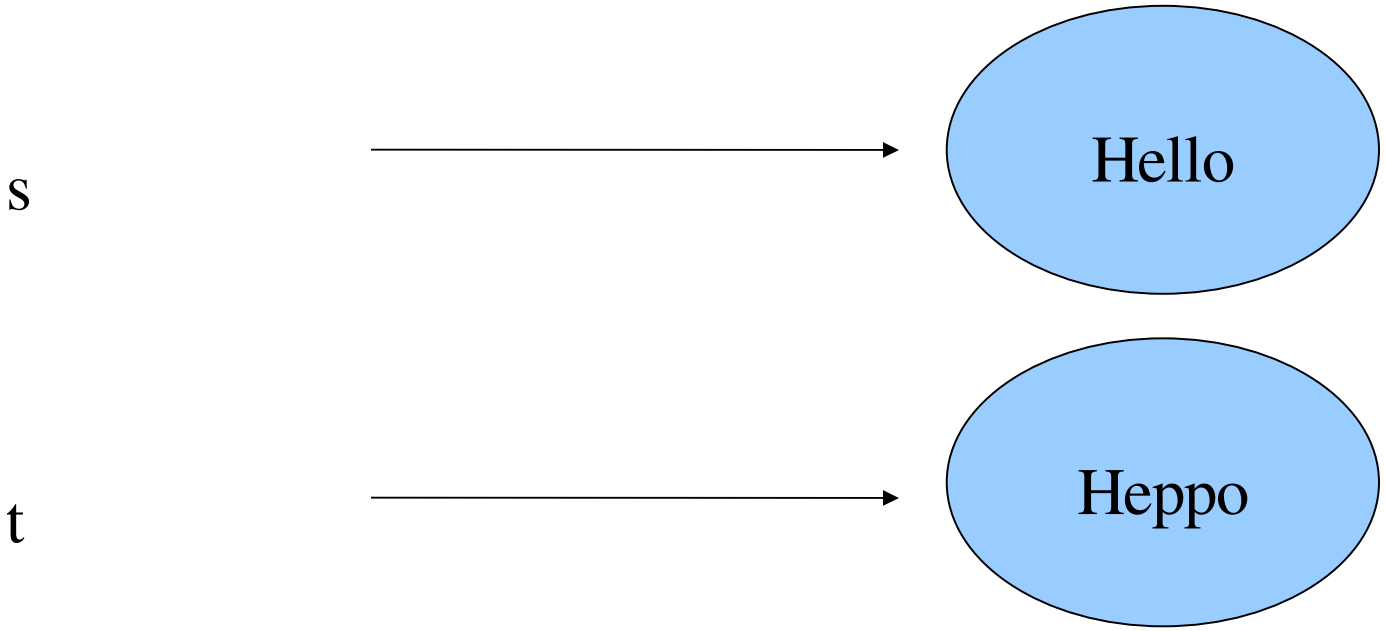
Strings

- Strings are immutable
- String methods
  - char charAt(int index)
  - String replace(char oldChar, char newChar)

Strings

String s = “Hello”

String t = s.replace('l', 'p');



Packages

- Classes in the standard Java library are grouped into packages
  - java.util
  - java.math
  - java .text

- Using the API

import

- To use a class from the library you must either use the fully-qualified name or import the class

```
java.util.Scanner s = new java.util.Scanner(System.in)
import java.util.Scanner;
import java.util.*; //import all classes in java.util.
```

Java Classes

- Contain data and methods
- Methods enable user to access and modify data

Encapsulation

- Data should be hidden from the user
  - Access to/modification of data is controlled by methods
- Modifiers
  - Enable programmer to specify which data and methods are private and which are public
    - private are accessible within the class
    - public are accessible from anywhere

Name.java

```
public class Name {
    private String first;
    private String last;

    //Constructor
    public Name(String thefirst, String thelast) {
                first =  thefirst;
                last = thelast;
    }

    public void printName() {
                System.out.println("First: " + first + " Last: " + last);
    }

}
```

Driver Classes

- The driver typically contains the main method
  - this is the starting point for the program
- The driver creates instances of other classes

NameTest.java

```
public class NameTest {
    public static void main(String[] args) {
            Name n = new Name("Sami", "Rollins");
            n.printName();
    }
}
```

Methods

```
public void printName() {
    ...
}
```

- *modifier return_type name(type name, ...)*
- You must specify the return type and the type of each parameter

More Method Headers

- For each data member, consider whether you should create *setter* and *getter* methods

public String getFirstName()

public void setFirstName(String newname)

public String getLastName()

public void setLastName(String newname)

Constructors

- Called when a new object is created
- Like a method with no return type
- Should initialize all relevant data
- Should take as input initial values for any variables that do not have a logical default value
- A default constructor takes no parameters

```
public Name(String thefirst, String thelast) {
    first = thefirst;
    last =  thelast;
}
```

Scope

```
public Name(String first, String last) {
    this.first = first;
    this.last =  last;
}
```

- this provide access to class variable
- first/last are local to the constructor