2. Variables, Objects, Classes

Aneesh Karve

Simple Java Program

```
public class Hello{
  public static void main(String[] args){
     System.out.println(“Hey dude.”);
  }
}
```

Variables

- type
- identifier (name)
- value

Variable Declaration (a.k.a. Definition)

- provide a name and type
  - `*type identifier;*`
  - `String slim;`

Variable Initialization

- provide a value for the first time
  - `variable = value;`
  - `slim = “shady”;`

Declaration with Initialization

- often occur simultaneously
  - *`type identifier = value;`*
  - `String audio = “slave”;`

Identifier Rules

- can only be made from
  - letters
  - digits
  - underscore
- cannot start with digits
- case sensitive
- cannot be reserved words

Identifier Conventions

- camelCase
  - `usedToSeparateWords`
- first letter lower case for variable names
- upper case for class names

Assignment Operator

- =
- works from *right to left*
- read as “becomes” or “refers to” so to avoid confusion with “equals”
- Examples
  - `int x = 31;`
  - `String name = “Bucky”;`

Numerical types

- integers (whole numbers)
  - `byte`
  - `short`
  - **`int`**
  - `long`
- floating point numbers (decimal numbers)
  - `float`
  - `double`
- all numerical types are *primitive types*

Arithmetic

- `double f = (a + b)/3.0;`

Class

- abstract concept
- user-defined type for objects
- defines data (variables) and behavior (methods)
- outside behavior (methods) = *public interface* which hides the *private implementation*

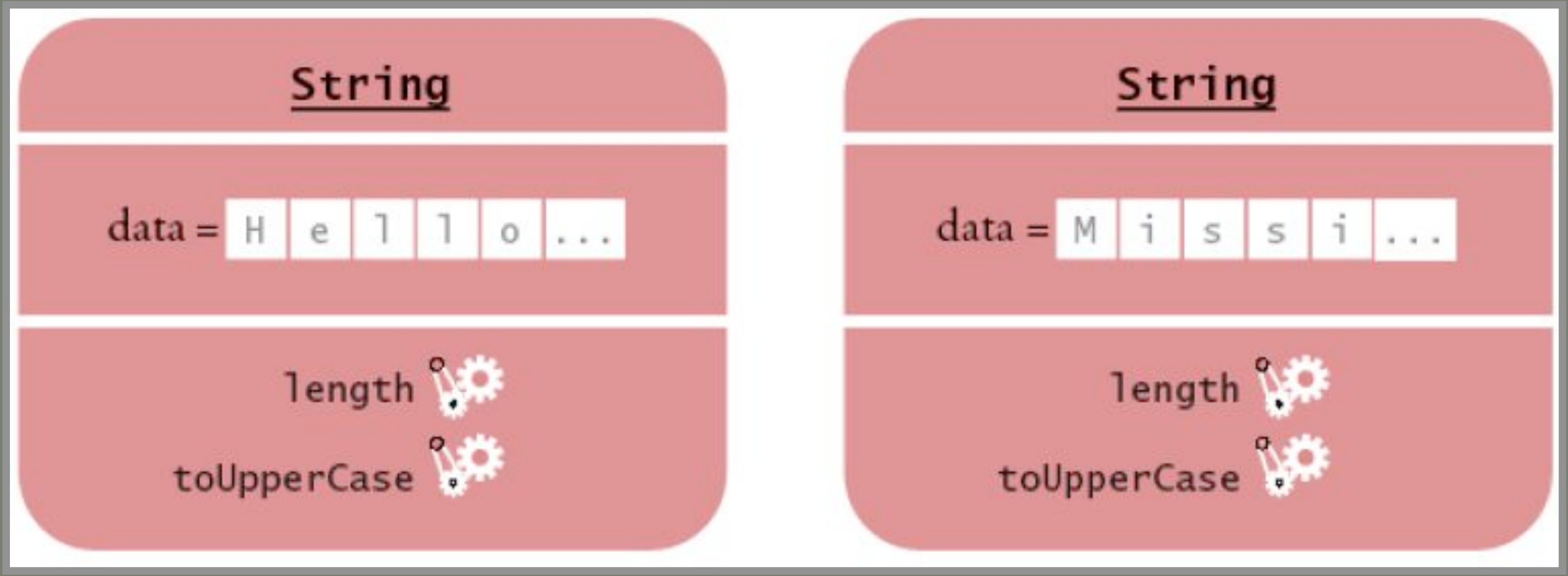
Uses of Classes

- *application class*
  - has main method
  - example: *tester class*
- *instantiable class*

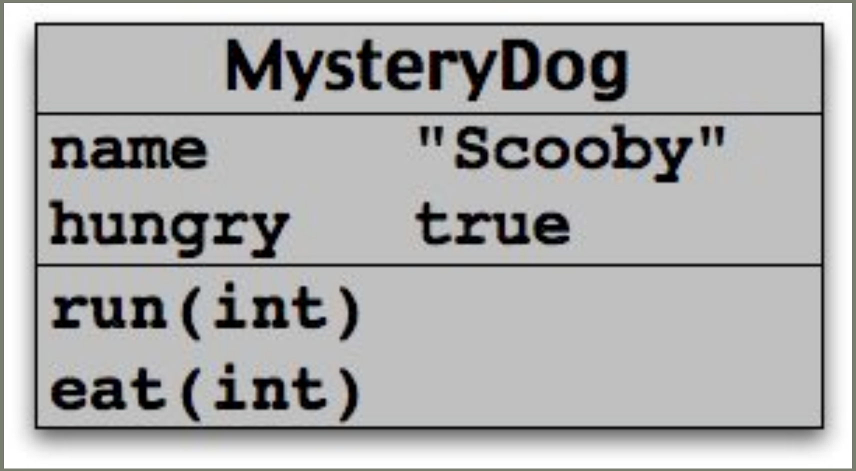
Object

- *instance* of a class (example of a concept, models a real “thing”)
- has data or *state* stored in its *instance fields*
- behaves or acts by *executing* methods defined by the class

String objects



Dot Operator



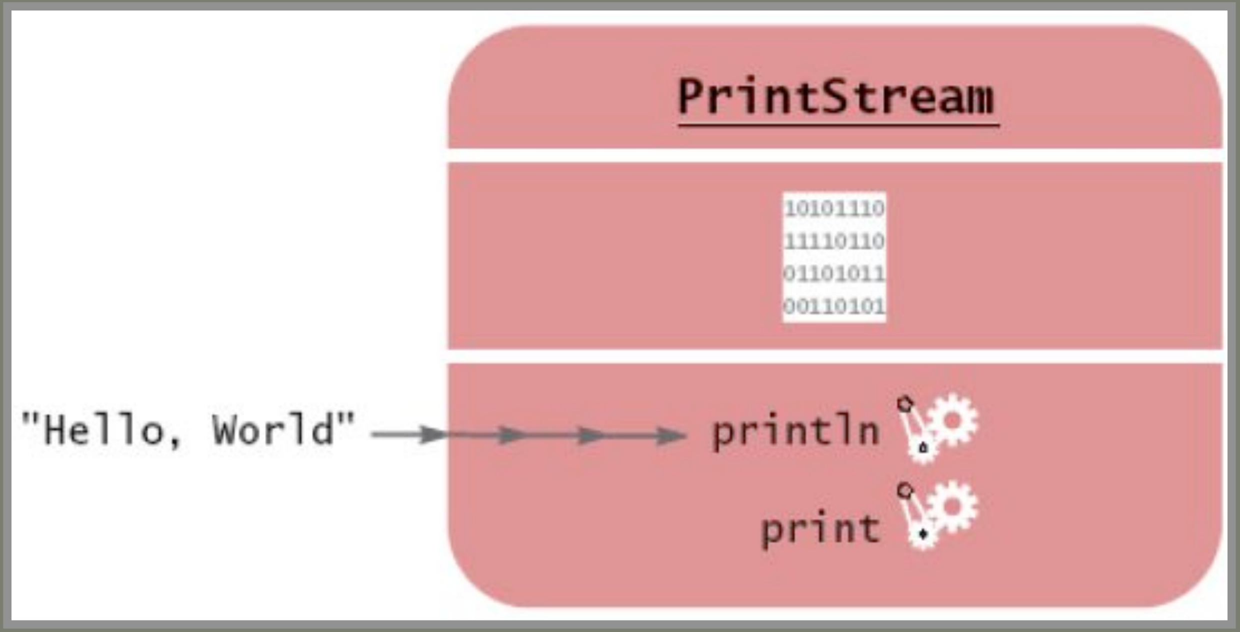
- Dot goes *inside* an object (to its methods or fields)
  - `scooby.eat(20);`
  - `scooby.hungry = false;`

Objects Construction

- new *ClassName(parameters, …);*
- *Rectangle r = new Rectangle(0, 0, 10, 10);*
- *r = new Rectangle();*

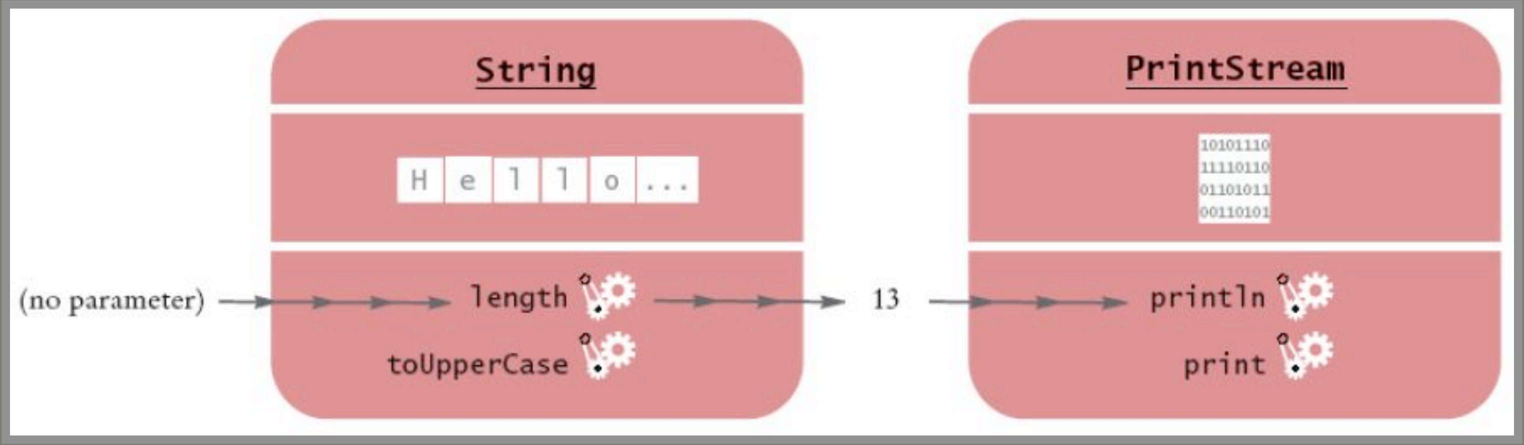
Parameters

- implicit
- explicit



Return values

- value returned by a method
  - `int numChars = slim.length();`



Accessor and Mutator Methods

- *accessor* - reports on state of implicit parameter
- *mutator* - *changes* state of implicit parameter

Primitive types

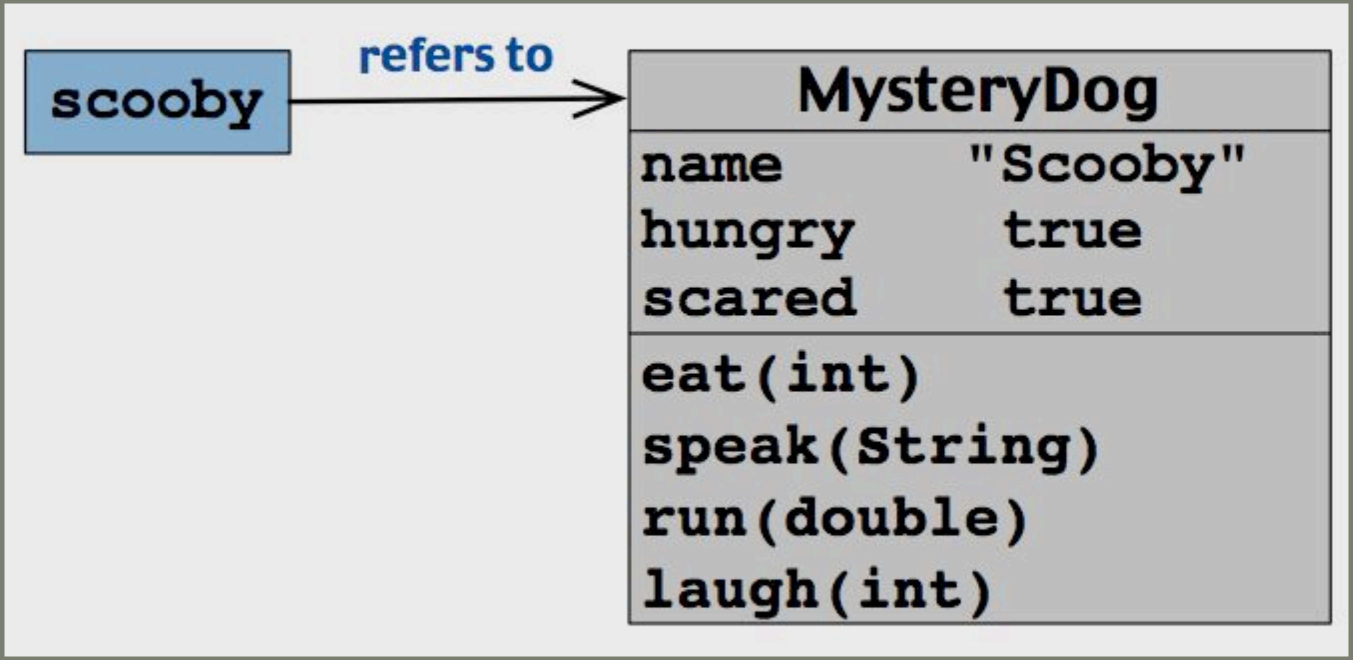
- a single value of fixed size and format
- these are all primitive types
  - numerical
    - `byte, short, int, long, float, double`
  - `char`
    - `‘a’`
    - `‘@’`
    - …
  - `boolean`
    - `true`
    - `false`

Object References

- *location* of an object in memory
- *aliasing* - multiple references point to same object

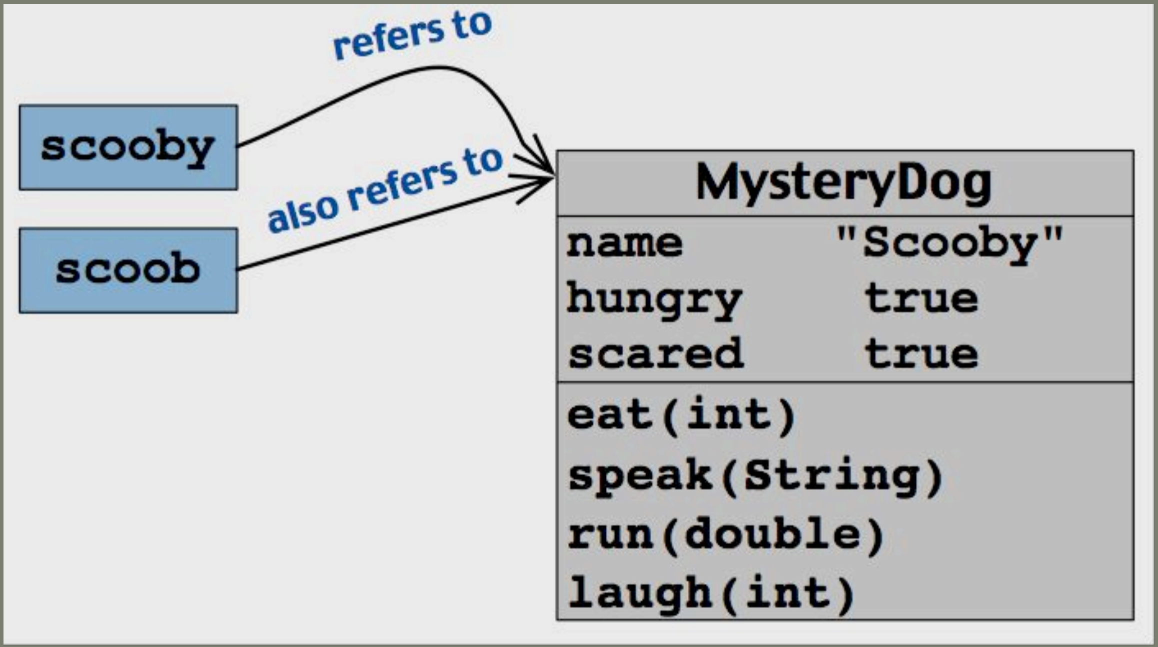
Object References

```
MysteryDog scooby = new MysteryDog();
```



Aliasing

```
MysteryDog scooby = new MysteryDog();
MysteryDog scoob = scooby;
```



Object references

```
String slim = “shady”;
String eminem = “marshall”;

slim = eminem;
int len = slim.length();

//what happens below?
System.out.println(len);
```

Primitive types don’t alias

```
int x = 44;
int y = x;
x = 55;
// y is still 44
// compare object references
```

Memory diagrams

Reference vs. Primitive Types

- Reference
  - values is an address or location for an object
  - assignment with ‘=‘ yields alias
- Primitive
  - value is an actual value
  - assignment with ‘=‘ yields true copy

`import`

- provides access to library classes
- occurs in first source lines of .java files
- `import java.awt.Rectangle;`
- `import java.awt.*;`

API documentation

- API - application programming interface
- J2SE API