CMSC 131

Object-Oriented Programming I

Introduction to Classes

Dept of Computer Science
University of Maryland College Park

This material is based on material provided by Ben Bederson, Bonnie Dorr, Fawzi Emad, David Mount, Jan Plane

Overview

- Objects
- Heap
- Equals

Objects

- Bundles of (related)
  - data ("state")
  - operations ("behavior")
- Data often referred to as instance variables
- Operations usually called methods
- Invoking operations can change state (values stored in instance variables)
- Example of objects
  - Bank Account
  - Student
  - Scanner
- Object-Oriented Programming
  - Program is a collection of interacting objects

Sample (Student Class)

State		Methods	
Name		getAge	date → age
ID		getGrades	sem., class → grades
DOB			
Major			etc.
	etc.		

Sample (Student Object)

State		Methods	
Name	Kerry Keenan	getAge	date → age
ID	444230695	getGrades	sem., class → grades
DOB	06-22-1987		
Major	CMSC		
	etc.		etc.

Classes

- Class → Blueprint/"Recipe" for objects
- Classes include specifications of
  - Instance variables (including types, etc.) to include in objects
  - Implementations of methods to include in objects
- Classes can include other information also, as will be seen later
  - Static methods / instance variables
  - public / private methods
  - And so on

Student Class Example

- Instance variables:

```
String name
int id
int dateOfBirth
String major
```

- Methods

```
getAge()
getGrades()
```

  etc.

- The actual class implementation will include code for the methods
- This describes a blueprint for student objects

Student Class (Definition Overview)

```
class Student {

    /* These are the instance variables */
    String name;
    int id;
    int dateOfBirth;
    String major;

    /* Instance methods */
    getAge() {
        // put code here
    }
    getGrades() {
        // put code here
    }

    Etc.
}
```

How Are Objects Created?

- In Java: using `new`

  Recall:

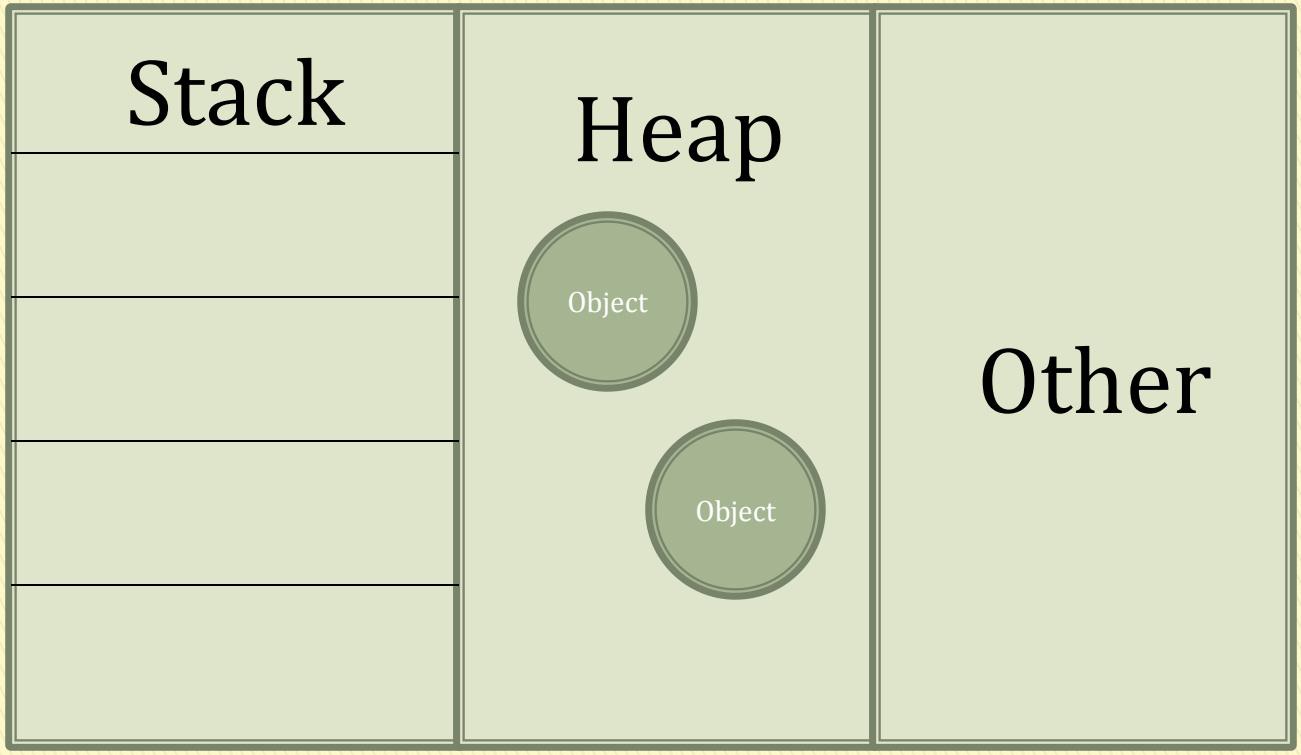
  ```
  Scanner sc = new
  Scanner(System.in);
  ```

- Invoking new:
  - Creates an object in a memory area called the "heap". Space is created for instance variables
  - Returns the address/reference where the object lives

Accessing State/Methods

- If
  - **`obj`** is an object reference
  - **`v`** is an instance variable of the object
  - **`m`** is a method of the object
- Then
  - **`obj.v`** is how to access the data `v` in **`obj`**
  - **`obj.m()`** is how to invoke `m` in **`obj`**
- So
  - If you have already done `String str = "Joe"`
- Then `str` is a String
  - `str` is an instance of a class.
  - Methods of this object→ `equals`, `compareTo`, etc.
  - `str.equals()`, `str.compareTo()`, etc. invokes these methods on that object

Main Memory Organization



In Java, 9 Sorts of Variables

- 8 primitive types
  - Types are the 8 built-ins (int, byte, double, etc.)
- Reference type
  - Objects always stored in heap (including all data)
  - Reference to objects are another type, and hold one memory address (typically one word)
- Stack holds local variables
  - e.g. `int x`
  - e.g. `String str; //` str is reference variable
- Heap holds allocated memory (i.e., with "`new`")
  - e.g. `Scanner sc = new Scanner(System.in);`

Strings Are Objects

- Where is `new` in

  `String name = "Batman";` ?
- Java provides it!
  - `String` is special because it is used so often
  - Java automatically "fills in" `new`
  - You can too:

    `String name = new String("Batman");`

Heap Issues

- What happens if new is called and there is no free heap?

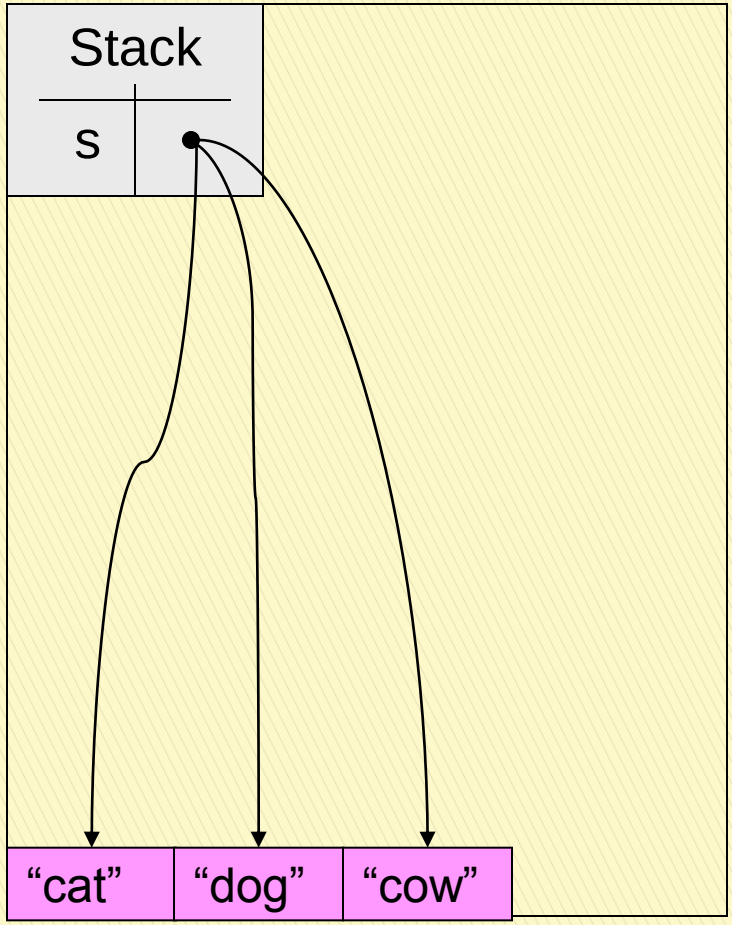
  Crash!
- What happens if following are executed?

  String s;

  s = new String(“cat");

  s = new String(“dog");

  s = new String(“cow");
- Wasted heap
  - “cat”, “dog” no longer referenced by stack
  - Crashes become a problem!



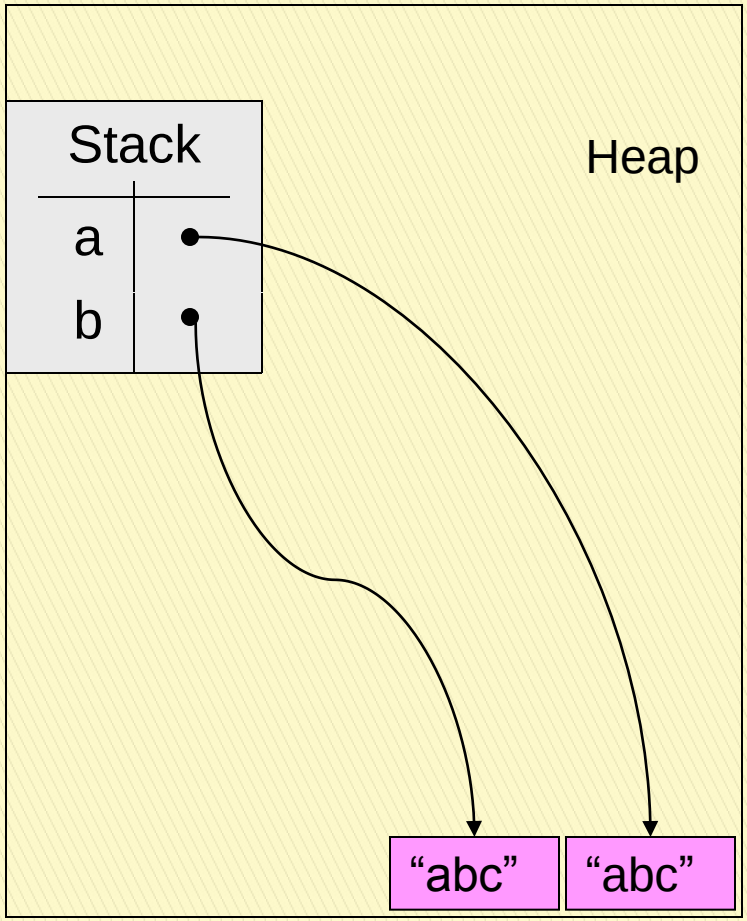
Garbage Collection

- This “heap management” or “memory management” issue is central in CS
- Java copes by invoking garbage collector to reclaim unused but still-allocated heap space
- Garbage collector reclaims memory in allocated heap and returns it to free heap
- In previous example, “cat” and “dog” would be reclaimed
- In some languages (e.g., C++) you need to take care of reclaiming memory
  - Use of delete operator in C++ otherwise you will have **memory leaks**

Example

```
String a = new String
  (“abc”);
String b = new String
  (“abc”);
if (a == b) {
  println (“Equal”);
} else {
  println (“Not equal”);
}
```

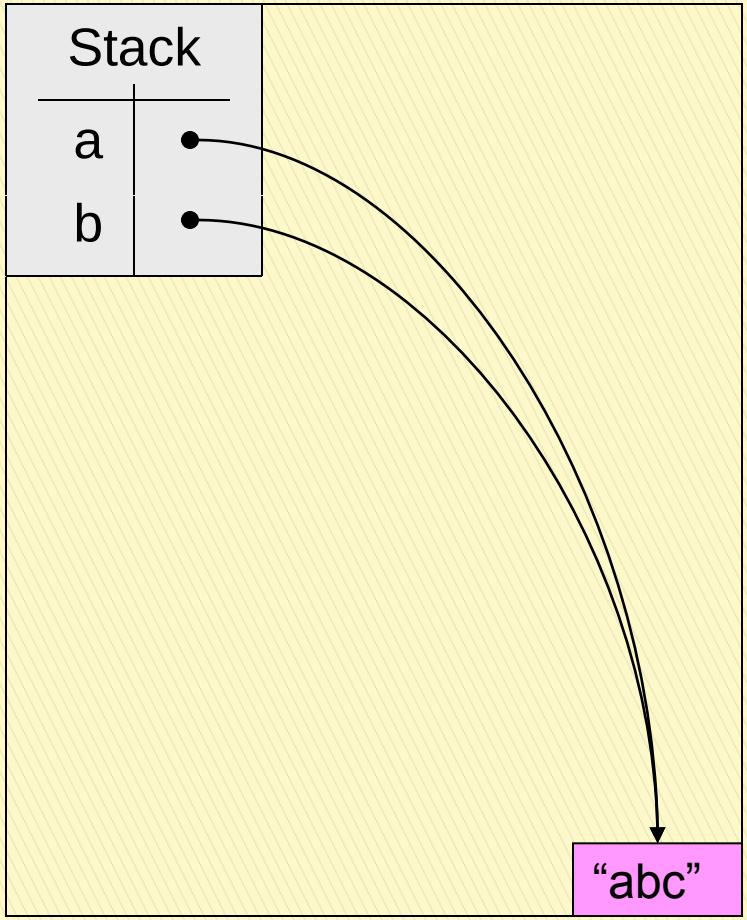
=> Not equal



Contrasting Example

```
String a = new String (“abc”);
String b = a;
if (a == b){
    println (“Equal”);
} else {
    println (“Not equal”);
}
    => Equal
```

- This is called ALIASING →Two variables refer to same object.



equals

- **==** checks if two reference variables refer to the same object
- Methods like `str.equals()` check if two different objects have the same "content"
- Other classes will have an `equals` method also

Example

- Let's define a class called SuperHero with
  - Instance variables name and strength
  - Get/Set methods
  - print method
- Let's define a driver class for our example
- Eclipse allow us to generate code ☺
  - Source → Generate Getters and Setters

Honors Material

- Here is a more advanced diagram for memory organization
  - http://blog.codecentric.de/wp-content/uploads/2009/12/java-memory-architecture.jpg