First name (color-in initial)	A	B	C	D	E	F	G	H	I	J	K	L	M	N	O	P	Q	R	S	T	U	V	W	X	Y	Z	section (9,10,or 11)	first name initial	last name initial
Last name (color-in initial)	A	B	C	D	E	F	G	H	I	J	K	L	M	N	O	P	Q	R	S	T	U	V	W	X	Y	Z			

H05: Due Monday 04.16.2012 in Lecture. Total Points: 50

MAY ONLY BE TURNED IN DURING Lecture ON Monday 04.16.2012, or offered in person, for in person grading, during instructor or TAs office hours.

See the course syllabus at https://foo.cs.ucsb.edu/56wiki/index.php/S12:Syllabus for more details.

---

(1) (10 pts) Fill in the information below. Also, fill in the A-Z header by

- **coloring in** the first letter of your first and last name (as it would appears in Gauchospace),
- writing **either 9,10,11** to indicate your **discussion section** meeting time
- writing your **first and last initial** in large capital letters.

All of this helps us to manage the avalanche of paper that results from the daily homework.

name:	
umail address:	@umail.ucsb.edu

Reading Assignment:

We are moving right along, reading three more chapters. The first two chapters are short, and a good bit of this is basic review of OOP concepts you may have already seen in CS24 (and possibly in CS32 if you took that, which is recommended, though not required, as a pre-req to CS56). The third describes a **major difference** between C++ and Java: the issue of **garbage collection**. This is a *crucial* chapter, so read it *carefully*.

- HFJ, Chapter 7, **165** through 196 and reading notes HFJ:Chapter _7
- HFJ, Chapter 8, **197** through 235. HFJ:Chapter _8
- HFJ:Chapter_9, **235** Life and Death of an Object (Constructors)
- If there are reading notes on the wiki, consult those too—sometimes they contain helpful hints.

---

(2) (6 pts) Based on your reading in HFJ Chapter 7:

Complete the following exercise from p. 179, putting a check next to the relationships that make sense.

	Oven extends Kitchen
	Guitar extends Instrument
	Person extends Employee
	Ferrari extends Engine
	FriedEgg extends Food

	Beagle extends Pet
	Container extends Jar
	Metal extends Titanium
	GratefulDead extends Band
	Blonde extends Smart
	Beverage extends Martini

(4) (5 pts) Based on your reading in HFJ Chapter 7:

What does it mean to have a "polymorphic argument" or a "polymorphic return type" for a method? Explain with an example—but NOT using the example of Vets and Animals used in the book. Substitute your own example. Give a detailed enough description of the class hierarchy you have in mind to make it clear that you get the concept.

(5) (5 pts) Based on your reading in HFJ Chapter 8:

Briefly describe the difference between an abstract class and an interface.

(8) (5 pts) There is code on p. 103 for a class to "test" the first iteration of the "sink a dot com" game developed in Chapter 5. Rewrite this as a class that uses JUnit testing (as illustrated in lab02).

If necessary, read: http://junit.sourceforge.net/doc/cookbook/cookbook.htm and/or refer to: http://junit.sourceforge.net/doc/faq/faq.htm

(9) (3 pts) Under what conditions does the compiler create a no-arg constructor for you?

(10) (3 pts) Under what conditions does the compiler NOT create a no-arg constructor for you?

(11) (5 pts) Given the following code excerpts:

```
public class Person {
   private String name;
   public Person (String name) {this.name = name;}
   public String getName() { return this.name;}
}
```

Write a class for Student that extends Person. Include a private attribute perm of type int. Include a constructor with the following signature:

```
public Student(String name, int perm) { ...
```

Use the proper technique (pp. 250-257) for invoking the parent class constructor (with a parameter) to initialize the name attribute.

(12) (8 pts) Based on what you learned from Chapter 9: Write a Java class that will compile and run (i.e. it needs a main() method) that has (ateast) the following four variables: a, b, c, and d, each instance of which will have the properties indicated. The class doesn't have to do any useful work---it is only to illustrate that you understand these concepts.

- a should be a primitive variable that will be stored on the stack
- b should be an object reference that will be stored on the stack (note: the references is on the stack, even though the object it refers to will always be on the Heap in Java.)
- c should be a primitive variable that will always be stored on the heap.
- d should be an object reference that will always be stored on the heap (note: here I want the reference variable itself to be on the heap, not just the object it refers to.)