Reference

Midterm Exam

CMSC 498i – Spring 2010

INSTRUCTIONS

This exam contains 9 pages, including this one. Write your name on the top of each before beginning. If you finish with more than 15 minutes left in class, you may bring your exam to the front and leave quietly. Otherwise, out of courtesy to others, turn your exam over and remain seated until the end. Some questions will ask you to write code. Don't worry if you syntax isn't perfect, what's important is conveying an understanding of the concepts.

When answering questions, write down any assumptions you make.

If you have a question, raise your hand.

If you don't have enough space to answer a question, use the back of the page, clearly numbering the question being answered. If you still need extra space, raise your hand to get more paper.

Good Luck!

QUESTION #1 — 20 POINTS

1.1 (4 points) – Inspect the following code. When done executing what is the employee object's retain count 2

```
NSMutableArray *array = [[NSMutableArray alloc] init];

Employee *employee = [[Employee alloc] initWithFullname:@"Sean Payton" employeeNumber:20];

[array addObject:employee];
```

1.2 (4 points) – Which of the following statements regarding Memory Management are true? Circle each.

~~a.~~ `+alloc` creates memory for an object and returns an object with a retain count of 0

~~b.~~ `-init` methods increment the retain count of objects created by +alloc

c. `-retain` increases the retain count of an object by 1

d. `-release` decreases the retain count of an object by 1

1.3 (4 points) – Explain the difference between the -release and -autorelease methods. Why is -autorelease needed?

- release decrements the retain count immediately, possibly causing immediate deallocation of the obj.
- autorelease leads to the retain count eventually being decremented (but not immediately). Specifically each autorelease "schedules" a -release message to be sent by putting the object into an autorelease pool.
- autorelease allows us to return objects to callers w/o requiring callers to release the object. w/o autorelease, this is not possible

1.4 (8 points) – Given the following code, find and fix the memory management errors.

```
@implementation Employee

- (NSDictionary *)dictionaryRepresentation {

    NSMutableDictionary *rep = [[NSMutableDictionary alloc] init];

    [rep setObject:[NSNumber numberWithBool:[self isSalaried]] forKey:@"isSalaried"];

    [rep setObject:[[self fullName] copy] forKey:@"fullName"];

    [rep setObject:[[self manager] dictionaryRepresentation] forKey:@"manager"];

    [rep setObject:[NSNumber numberWithInt:[self employeeNumber] forKey:@"employeeNumber"]];

    return rep;
}

@end
```

→ -copy leads to a leak!

also acceptable would be [[self fullName] copy] autorelease]

return [rep autorelease];

```
@implementation Team

- (Employee *)coach {

    Employee *owner = [[Employee alloc] initWithFullname:@"Tom Benson" employeeNumber:1];

    [owner setIsSalaried:YES];

    Employee *coach = [[Employee alloc] initWithFullname:@"Sean Payton" employeeNumber:20];

    [coach setIsSalaried:YES];

    [coach setManager:owner];

    [owner release]; // since its now retained by the coach

    return coach;

}
@end
```

QUESTION #2 — 20 POINTS

2.1 (15 points) – Write the Employee.h declarations that provide the functionality needed by Question #1.4.

- You must declare any necessary instance variables
- You must use method syntax for everything (do not use property syntax)
- Only declare what is needed. e.g. The employee's `fullName` can be accessed but not changed after init

```
@interface Employee : NSObject {
@private
    NSString *fullName_;
    int employeeNum_;
    BOOL isSalaried_;
    Employee *manager_;
}

- (id) initWithFullName:(NSString *)fullName employeeNumber:(int)empNum;
- (NSString*) fullName;
- (int) employeeNumber;

- (void) setIsSalaried:(BOOL)isSal;
- (BOOL) isSalaried;

- (void) setManager:(Employee *)mgr;
- (Employee *) manager;

- (NSDictionary *) dictionaryRepresentation;

@end
```

2.2 (5 points) – Use Objective-C 2.0's property syntax to declare a "nickname" property for the `Employee` class. Your declarations should support the following usage:

```
[employee setNickname:@"Tommy"];
NSLog(@"Hi %@", [employee nickname]);
```

```
@property (copy, readwrite) NSString *nickname;
```

QUESTION #3 — 20 POINTS

3.1 (15 points) – Implement the following methods for the `Employee` class (you may not use property synthesis)
– The "init", `setIsSalaried:`, and `copyWithZone:` methods

```
- (id) initWithFullName:(NSString *)fn employeeNumber:(int)enumb{
    if ((self = [super init])) {
        fullname_ = [fn copy];
        employeeNumber_ = enumb;
    }
    return self;
}

-(void) setIsSalaried:(BOOL) isSal {
    isSalaried_ = isSal;
}

-(id) copyWithZone:(NSZone*)z {
    Employee *c = [[[self class] alloc] initWithFullName:[self fullName]
                                          employeeNumber:[self employeeNum]

    [c setIsSalaried:[self isSalaried]]
    [c setManager:[self manager]]
    return c;
}
```

3.2 (5 points) – Explain Objective-C 2.0's property synthesis feature and demostrate its use. Implement the `fullName` property for the `Employee` class using property synthesis syntax.

property synthesis essentially tells the compiler to auto-generate setters and getters for you. You can tell the compiler which ivar to use for storage with @synthesize, and tell the compiler how to implement the methods using attributes like nonatomic, copy, retain in the @property line.

```
@synthesize fullName = fullName_;
```

QUESTION #4 — 10 POINTS

Consider an application that displays a company's organizational structure and allows managers to modify the reporting structure by dragging images in a view. Managers are allowed to only modify the structure below them. For example, they can't make the CEO report to them. Imagine the organizational view is implemented by a UIView that does not want to know anything about the current user or their credentials.

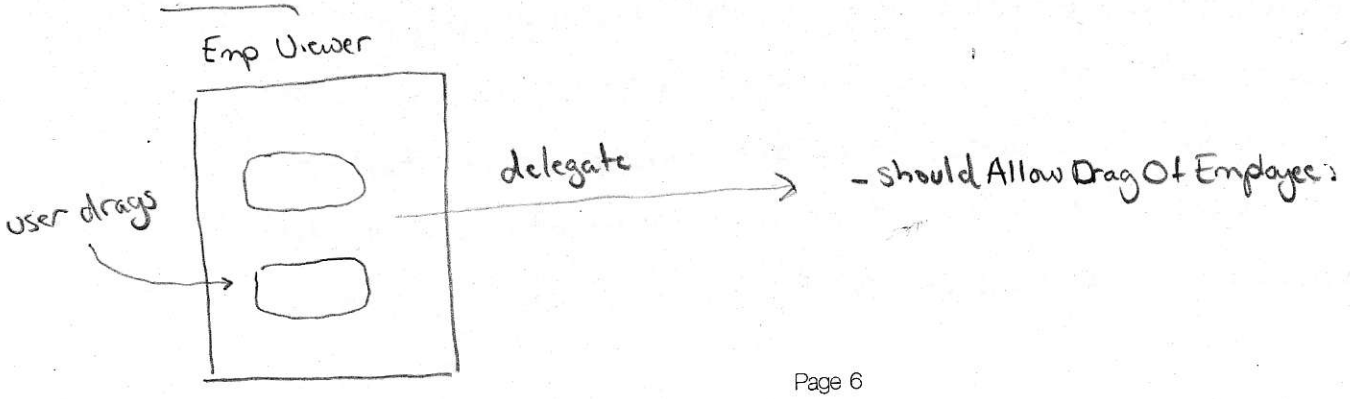
When deciding whether the user is allowed to drag an employee to a new reporting manager which design pattern would be more appropriate to use, delegation or notification? Explain why.

delegation. delegates are used to control policy and customize beh.

—

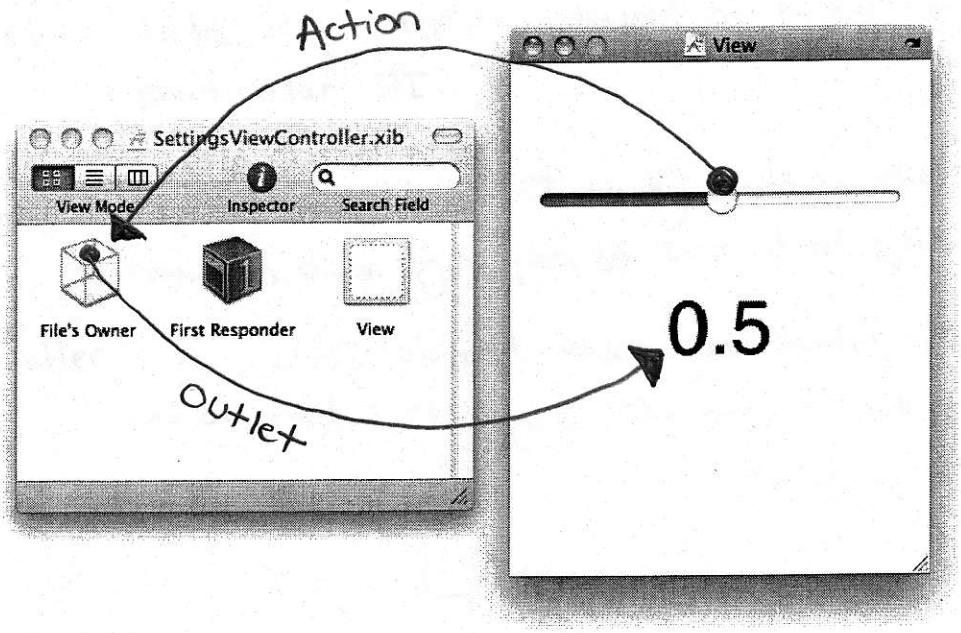
notifications are used to broadcast information, and are one-to-many. Given that there are multiple recipients notifications aren't good for "should" type of messages. In fact, NSNotifications don't provide any way to return a value.

delegation provides a one-to-one connection for sending messages and allows a returned value. So, in addition to the "will" and "did" messages a notif observer would get, delegates also get "should" messages. delegates are often used to customize object behavior or determine policy.



QUESTION #5 — 10 POINTS

Consider the following XIB file screenshot. Assume you are implementing a SettingsViewController class that must update the UILabel to display the current value of a UISlider's value whenever it changes.



5.1 (5 points) – What IBAction and IBOutlets do you need defined in your SettingsViewController. Draw an arrow on the XIB screenshot that shows the location and interaction of your IBActions and IBOutlets. Label each arrow with either IBOutlet or IBAction.

5.2 (5 points) – What is the File's Owner? Why is the File's Owner construct used?

in this example it is the SettingsViewController. In general, it simply represents the already allocated object passed to the nib loader as the "owner" parameter. The important point is that nib loading does not create the object, but we still are afforded access to it in the XIB. This allows us to make connections to/from these sort of external objects in IB

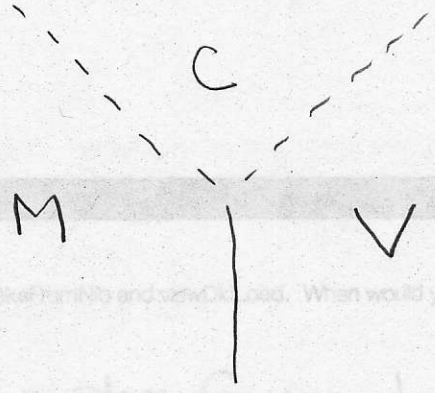
QUESTION #6 — 10 POINTS

Describe the model–view–controller architecture including a discussion of its benefits. Provide a short description (no more than 3-4 sentences) of each of Model, View and Controller.

Model - data only objects. devoid of knowledge of any particular UI.

View - display only object. Typically deals in the lowest common data type possible instead of specific models.

Controller - Mediating object that coordinates display updates and model changes. The only object that communicates directly with the others.



Benefits

- Clean seperation leads to many benefits
- Cleaner, less complex code in each
- Easier to unit test
- Code (M, V) typically more reusable
- Easier to change code in one part of system w/o affecting others.

QUESTION #7 — 5 POINTS

Describe the relationship between UIView's -layoutSubviews method and ~~-setNeedsDisplay.~~ -setNeedsLayout

-layoutSubviews is where you put code that positions subviews. You should not invoke it directly

-setNeedsLayout schedules -layoutSubviews to be called at some point before the next display. Although you may call it N times -layoutSubviews will still only be called 1 time per display.

QUESTION #8 — 5 POINTS

Explain the difference between awakeFromNib and viewDidLoad. When would you use one vs. the other?

Each method is a place for you to put post-nib loading setup code. each is invoked *after* outlet and action nib connections are setup (ie. outlets are no longer *nil*)

~

awakeFromNib is sent to object created during nib loading and not to already existing items like the Files owner.

ViewControllers are often files owner and therefore do not get -awakeFrom instead, UIViewController provides -viewDidLoad, a method it will call after the nib loading code returns.