Midterm 1
COMP 401
Spring 2012

I have not received nor given any unauthorized assistance in completing this exam.

Signature: ______________________

Name: _________________________

PID: ___________________________
This page intentionally left blank.
Part I: Multiple Choice and Short Answer

15 questions, 2 points each, 30 points total.

NOTE: For each multiple choice question, more than one answer may be correct.

I.1) Which of these is not a valid primitive data type?

a) int
   * b) unsigned short
   c) char
   d) double
   * e) true

Java does not have “unsigned” integer types.
True is a valid boolean value but not a type.

I.2) Which of these is not a legal expression?

a) 3 * 2 + 7 - 11
   * b) "hello".length() + "5"
   c) true && (3 + 4)
   d) false || (3 > 5)
   e) "hello".charAt(4) == 'h'

The && operator expects a boolean value on either side of it. The expression (3+4) is not boolean. Choice b is valid because the + operator is able to deal with integers and strings together (it converts the integer back to a string).

I.3) Which of these is not a legal variable name?

a) fourEleven
b) FOUR_ELEVEN
   * c) 4eleven
   d) for_11
   * e) _four eleven_

C is not legal because it starts with a digit. E is not legal because it contains a space.

I.4) Which of these is not true of statements and blocks in Java?

a) A statement must end in a semicolon.
   * b) A statement must result in a value.
   c) A block of statements is enclosed in curly braces.
   d) A block of statements can be placed wherever a single statement is valid.
   e) A block of statements can be empty (i.e., have no statements in it).

A statement does not have to produce a value, but can just have a side effect (for example, a method call with a void return signature.)
I.5) What will the following code print?

```java
int sum = 0;
for (int i = 0; i < 3; i++) {
    sum += i;
}
System.out.println(i);
```

a) 0  
b) 6  
c) 3  
d) 2  
* e) The code will not run because there is an error.

I.6) What is the value of `bar` after the following code executes?

```java
int foo = 72;
int bar = 0;

if (foo < 90) {
    bar = 10;
}
if (foo < 80) {
    bar += 5;
} else if (foo > 70) {
    bar -= 5;
}
```

Answer: _____ 15 _______

I.7) Which of these statements are correctly related to the principle of encapsulation?

a) Object state should be exposed via public instance fields.  
* b) An object encapsulates its state by preventing direct access to its instance fields.  
c) Every object property must be associated with both getter and setter methods.  
* d) Object properties are exposed via public getter and setter methods.  
* e) Setter methods should validate values if possible.

A is the opposite of encapsulation and C is not true for read-only properties.
I.8) Which of these statements are correctly related to the factory design pattern?

a) The factory design pattern is useful for collections. 
* b) The factory design pattern is useful when object construction requires complex validation. 
* c) A class using the factory design pattern should have private constructors. 
* d) A class using the factory design pattern will provide one or more class methods that return instances of the class. 
e) A class using the factory design pattern will not need a constructor. 

A is not true because factory design pattern does not have anything to do with collections. E is not true because the factory method will still need a constructor in order to create new objects.

I.9) Which of these statements are not true of interfaces?

a) Interfaces are defined by a set of method signatures. 
* b) Interface names are not valid data types for variables. 
* c) Interfaces should provide a default implementation for each specified method. 
d) Interfaces can be extended using inheritance. 
* e) Interfaces can specify one or more constructors.

Interface names are used as data types for variables that reference objects that implement the interface. Interfaces do not have any implementation code or constructors, just method signatures.

I.10) Which of these statements is true of public class methods?

a) Instance fields are available through the this keyword. 
* b) Class methods can be called within instance methods. 
c) Class methods require an instance of the class to be created. 
* d) Class methods can be called via the dereferencing operator using the class name. 
e) Class methods should not be public.

Class methods can always be called within the class. Similarly, D is true because public class methods are called from outside the class using the class name and the dereferencing operator.

I.11) What is the value of the variable bar after the following code executes?

```java
int bar;
int[] a = {1, 2, 3, 4, 5};
int[] b = a;
b[2] = a[1];
bar = a[2];
```

When b is set equal to a, they are pointing to the same array. So when b[2] is set to a[1], this also sets a[2] since a and b are the same array.

Answer: ____ 2 _________
I.12) Which of these statements is not true of the `String` data type?

* a) `String` is equivalent to `char[]`.  
  b) Strings are immutable.  
* c) The `==` operator provides a character-by-character comparison of two `String` values.  
  d) The `+` operator can be applied to a `String` and an `integer`.  
  e) `String` is a subclass of `Object`.  

Strings are not just arrays of characters but an object type that represents an immutable sequence of characters. The `==` operator only checks if two strings are the same object (i.e., same place in memory). The method `equals()` checks by value of the characters themselves.

I.13) Which of these statements properly declares a named constant?

a) `static public int A_CONSTANT = 3;`  
   * d) `static public final int A_CONSTANT = 3;`  
   b) `static public final int A_CONSTANT;`  
   c) `public final int A_CONSTANT = 3;`  
   * e) `static final public int A_CONSTANT = 3;`  

Named constants are class fields so require the “static” keyword, they are constant and thus require the “final” keyword. The order of these keywords does not matter.

I.14) What will the following code print?

```java
int value = 2;

switch(value) {
    case 0:
        System.out.println("A");
    case 1:
        System.out.println("B");
    case 2:
        System.out.println("C");
    case 3:
        System.out.println("D");
    default:
        System.out.println("E");
}
```

Answer: C

D

E

Because there is no `break` statement, execution continues all the way through the switch statement after starting at the appropriate case.
I.15) Which of the following statements is not true of objects?

* a) Every object is associated with at least one interface.
b) The state of an object is determined by the current values of its instance fields.
c) Any variable that holds an object reference can be set to null.
* d) A new copy of an object is created when it is passed to a method.
e) Two object references that have the same value are referring to the same object.

Objects may implement an interface but are not required to. Objects are passed by reference value and not copied.
Part II: Identifying code errors.

The following code is meant to define an interface called Media and a class called Song that implements Media. Six lines in the definition of Song contain errors. Circle these lines and provide corrected versions of these lines to the right.

2 points per error, 12 points total.

```java
public interface Media {
    String getName();
    double getLength();
    int getRating();
    void setRating(int new_rating);
}

public class Song extends implements Media {

    private static String name;
    private double length;
    private int rating;

    public Song(String name, double length, int rating) {
        this.name = name;
        this.length = length;
        this.rating = rating;
    }

    public String getName() {
        return name;
    }

    public double getLength() {
        return length;
    }

    public int getRating() {
        return rating;
    }

    public void setRating(int rating) {
        this.rating = this.rating;
    }
}
```
Part III: Understanding Code Structure

The questions in this section rely on code that defines two classes: Card and Deck. These classes are intended to represent standard playing cards organized into 4 suits with 13 ranks. The text of two class files, Card.java and Deck.java, that define these two classes are given below. The questions for Part III follow.

Card.java:

```java
package game;

public class Card {
    public enum Suit {SPADE, HEART, DIAMOND, CLUB};
    
    public static final Suit[] SUITS = {
        Suit.SPADE,
        Suit.HEART,
        Suit.DIAMOND,
        Suit.CLUB};

    public static final int ACE = 1;
    public static final int KING = 13;
    public static final int QUEEN = 12;
    public static final int JACK = 11;

    private Suit suit;
    private int value;

    public Card (Suit suit, int value) {
        this.suit = suit;
        this.value = value;
    }

    public Suit getsuit() {
        return suit;
    }

    public int getValue() {
        return value;
    }
```
Card.java continued:

```java
public boolean equals(Object o) {
    Card other_card = (Card) o;

    if ((other_card.suit == suit) &&
        (other_card.value == value)) {
        return true;
    }
    return false;
}

public int compare(Card other) {
    if (other.value > value) {
        return -1;
    } else if (other.value < value) {
        return 1;
    }
    return 0;
}
}
```

Deck.java:

```java
package game;

import java.util.ArrayList;
import java.util.Random;
import java.util.Iterator;

public class Deck implements Iterable<Card> {
    private ArrayList<Card> cards;

    public Deck(ArrayList<Card> cards) {
        this.cards = cards;
    }
}
```
public static Deck makeFullDeck() {
    ArrayList<Card> cards = new ArrayList<Card>();

    for (int value = 1; value <= 13; value++) {
        for (Card.Suit s : Card.SUITS) {
            cards.add(new Card(s, value));
        }
    }

    return new Deck(cards);
}

public int size() {
    return cards.size();
}

public Card getCardAt(int position) {
    return cards.get(position);
}

public Iterator<Card> iterator() {
    return cards.iterator();
}

public Deck splitDeck() {
    ArrayList<Card> mine = new ArrayList<Card>();
    ArrayList<Card> yours = new ArrayList<Card>();

    Iterator<Card> dealer = iterator();

    while (dealer.hasNext()) {
        mine.add(dealer.next());
        yours.add(dealer.next());
    }

    cards = mine;
    return new Deck(yours);
}
Deck.java continued:

```java
public void shuffle() {
    Random rng = new Random();
    for (int i=0; i<cards.size(); i++) {
        int swap_pos = rng.nextInt(cards.size());
        Card c = cards.get(i);
        cards.set(i, cards.get(swap_pos));
        cards.set(swap_pos, c);
    }
}
```

9 questions, 2 points each, 18 points total.

III.1) List the instance fields (if any) of the class Deck

cards

III.2) List the instance methods (if any) of the class Deck

size(), getCardAt(), iterator(), splitDeck(), shuffle()

III.3) List the class fields (if any) of the class Deck

There are none.

III.4) List the class methods (if any) of the class Deck

makeFullDeck()
III.5) How many constructors does the class Deck define?
1

III.6) What is the return type of the shuffle() method in the class Deck?
void

III.7) What is the return type of the getSuit() method in the class Card?
Card.Suit or Suit

III.8) What is the return type of the iterator() method in the class Deck?
Iterator<Card>

III.9) What package contains the class Random?
java.util
Part IV: Writing Code

This part relies on the definitions of Card and Deck provided in Part III.

3 questions, 5 points each, 15 points total.

IV.1) If we use the Card and Deck classes as defined above and execute the following code:

```java
Deck d1 = Deck.makeFullDeck();
System.out.println(d1.size());
Deck d2 = d1.splitDeck();
System.out.println(d2.size());
Deck d3 = d2.splitDeck();
System.out.println(d3.size());
Deck d4 = d3.splitDeck();
System.out.println(d4.size());
```

We will obtain the following output:

52
26
13
Exception in thread "main" java.util.NoSuchElementException

Clearly, we have an error somewhere in splitDeck(). Describe why the error occurs and provide a corrected version of splitDeck() below.

In the while loop, there are two consecutive calls to the iterator’s next() method without checking to see if the deck still have another card between those two calls. When the deck has an odd number of cards, this will cause an exception. Change these two lines to something like this:

```java
mine.add(dealer.next());
if (!dealer.hadNext()) {
    break;
}
yours.add(dealer.next());
```
IV.2) Suppose we wanted to add a new read/write property for Card objects that indicates if the card was "face up" or "face down". We would like this new property to follow JavaBeans conventions. Provide any additional code that you would need to add to the class definition. Assume that the following enumeration defines the symbols used to represent "face up" and "face down".

```java
public enum CardStatus {FACE_UP, FACE_DOWN};

private CardStatus status;

public CardStatus getStatus() {
    return status;
}

public void setStatus(CardStatus status) {
    this.status = status;
}
```
IV.3) The Card class defines a comparison method with the following signature:

public int compare(Card other)

This method returns -1 if the other card (i.e., the one passed to this method as a parameter) is higher in value, 1 if the other card is lower in value, and 0 if the other card has the same value. The suits of the cards are not consider. This means that the Ace is considered the lowest card in the deck. Provide a new version of the compare method below that takes an additional boolean argument indicating whether the Ace should be considered the highest or lowest card. The method signature of this method should be:

public int compare(Card other, boolean ace_is_high) {
    if (ace_is_high) {
        if (value == ACE) {
            if (other.getValue() == ACE) {
                return 0;
            }
            return 1;
        } else if (other.getValue() == ACE) {
            return -1;
        }
    }
    return compare(other);
}

Or anything logically equivalent.