Name: _____

# CS 141: Introduction to (Java) Programming: Exam 2 Solutions

*Jenny Orr • Willamette University • Fall 2017*

| | | | |
|---|---|---|---|
| 1. | (max 30) | 3. | (max 24) |
| 2. | (max 30) | 4. | (max 16) |
| Total: | | | (max 100) |

1. (1 pts each, 30 pts total) **True and False:** Please circle T or F *(Credit is only given if the instructor can clearly tell which answer is circled)*

   1) **T** **or** **F**: A class cannot have more than one superclass.

   2) **T** **or** **F** In an abstract class, not all of the methods are implemented.

   3) **T** **or** **F:** If one changes the value of a class variable, the value is changed for all objects of that type.

   4) **T** **or** **F:** If Electric is a subclass of Vehicle, then it is ok to have the declaration:

   ```
   Vehicle e = new Electric();
   ```

   5) **T** **or** **F:** An object can be created from a concrete class.

   6) **T** **or** **F:** The keyword `super` is used to call methods of the parent class.

   7) **T** **or** **F**: To create a subclass, one uses the `implements` keyword.

   8) **T** **or** **F**: A subclass has *direct* access to *private* instance variables of its superclass.

   9) **T** **or** **F:** Array parameters are passed (i.e. called) by value.

   10) **T** **or** **F**: A class inherits data and behavior from its subclass.

   11) **T** **or** **F:** The length of an array is immutable.

   12) **T** **or** **F:** Overriding is when methods in a class have the same name but a different list of parameters.

   13) **T** **or** **F:** ArrayList parameters are passed (i.e. called) by reference to a method.

   14) **T** **or** **F:** A protected member variable of a super class may be directly accessed by its subclass.

   15) **T** **or** **F:** If `Course` is an abstract class, then it is ok to have the declaration

   ```
   Course[] c = new Course[10];
   ```

   16) **T** **or** **F:** An example of polymorphism is when an array contains objects of different types but where those types are related through inheritance.

17) **T or F:** Overloading a method occurs when a method in the subclass has the same name and parameters as a method in the superclass.

18) **T or F:** An `actionPerformed` event is generated when a button is clicked.

19) **T or F:** The key word `this` is the name an object can use to refer to itself.

20) **T or F:** A class's static methods can access instance member variables.

21) **T or F:** The keyword word `static` is used to create instance member variables.

22) **T or F:** An object of a class does not have to be created in order to execute static methods of the class.

23) **T or F:** An exception is an object that is thrown when an error or an unexpected event occurs during runtime.

24) **T or F:** A `try` - `catch` clause is used to gracefully respond to exceptions.

25) **T or F:** A deep copy of an object creates new references but not necessarily new objects.

26) **T or F:** An object can be created from an interface.

27) **T or F:** All methods specified by an interface should be public.

28) **T or F:** A Java interface is used to force a concrete class to implement certain methods.

29) **T or F:** An interface has methods but no instance variables.

30) **T or F:** A class can implement multiple interfaces.

2. (6 pts each, 30 pts total) Arrays of primitives and objects:
   a. Write code that declares and creates a 1D array called `myCards` which consists of 100 Card *objects.* Use the default Card constructor.

```
Card[] myCards = new Card[100];

for (int i=0; i < myCards.length; i++) {
    myCards[i] = new Card();
}
```

b. Write a loop that prints the `myCards` array backwards.

```
for (int i = myCards.length-1; i >= 0; i--) {
    System.out.println(myCards[i]);
}
```

c. Write a loop that finds and prints the largest card index contained in the cards of the `myCards` array. Assume there is a `getIndex()` method.

```
int max = myCards[0].getIndex();
for (int i = 1; i < myCards.length; i++) {
    if (myCards[i].getIndex() > max) {
        max = myCards[i].getIndex();
    }
}
System.out.println("Max index is " + max);
```

d. Write code that creates a 2D array of doubles called `nums` which has 2 rows, where the 0th row contains 100 elements, and the next row contains 120 elements.

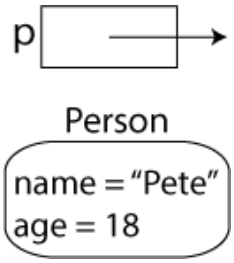
```
double[][] nums = new double[2][];
nums[0] = new double[100];
nums[1] = new double[120];
```

e. Write code that uses a nested loop to set the values in nums to random numbers. Make use of the length variable rather than explicitly using numbers such as 100.

```
for (int i = 0; i < nums.length; i++) {
    for (int j = 0; j < nums[i].length; j++) {
        nums[i][j] = Math.random();
    }
}
```

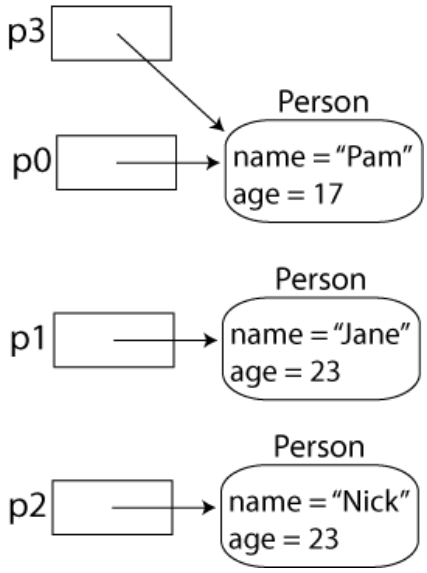
3. (22 pts total) Object Diagrams: Suppose you have a Person class which contains 2 member variables: the name and age of a person.

   Given the code below:




```
Line 1:     public static void main(String[] args) {
Line 2:       Person p0 = new Person("Pam",17);
Line 3:       Person p1 = new Person("Jane", 23);
Line 4:       Person p2 = new Person(p1);  // copy constructor
Line 5:       p2.setName("Nick");
Line 6:       Person p3 = p0;
Line 7:       System.out.println("p0: " + p0 + "\n p1: " + p1 +
                                 "\np2: " + p2 + "\n p3: " + p3);
Line 8:       Person[] ps = { p0, p1, p2, p3 };
Line 9:       ps[0]= ps[1];
Line 10:      ps[3]= new Person("Dan",10);
Line 11:      System.out.println(ps[0]+"\n"+ps[1]+"\n"+ps[2]+"\n"+ps[3]);
Line 12:    }
```

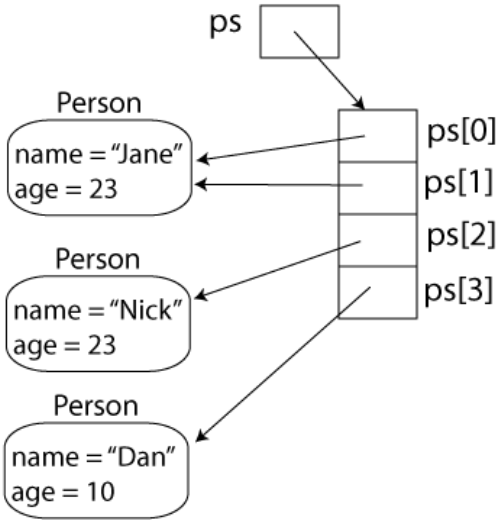
   a) (9 pts) Draw the object diagram for all references and objects as they exist after Lines 1-6 have been executed. Use the drawing style used in class, e.g. *rectangular boxes indicate object <u>references</u> and rounded boxes to indicate <u>objects</u>* as shown above on the right. Include the values of the objects' member variables as shown in the picture.




   b) (2 pts) Assuming the toString for the Person class prints the name and age in the form
      `Pete 18`
      What is printed at Line 7?

```
p0: Pam 17
p1: Jane 23
p2: Nick 23
p3: Pam 17
```

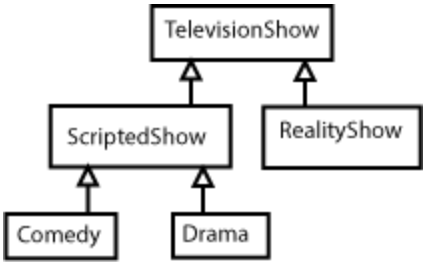
c) (9 pts) Draw the object diagram for the `ps` array (references and objects) as it exists after all lines have been executed.




d) (2 pts) What is printed at Line 11?

```
Jane 23
Jane 23
Nick 23
Dan 10
```

4. (16 pts total) Inheritance:
   a. (9 pts) For the class inheritance structure shown on the right (no interfaces are included):




      i. (2 pts) What (if any) is the superclass of ScriptedShow?
         Ans: **TelevisionShow**

      ii. (2 pts) What (if any) is the superclass of TelevisionShow?
         Ans: **Object**

      iii. (2 pts) What (if any) is a subclass of TelevisionShow?
         Ans: **ScriptedShow or RealityShow**

      iv. (3 pts) Which of the following are <u>valid</u> class headers that would be found in the above hierarchy? Circle all that apply.

```
a)  public class TelevisionShow extends RealityShow {. . .}
b)  public class TelevisionShow implements RealityShow {. . .}
c)  public class RealityShow extends TelevisionShow {. . .}
d)  public class Comedy implements ScriptedShow {. . .}
e)  public class Drama extends TelevisionShow {. . .}
f)  public class Comedy extends ScriptedShow {. . .}
```

   b. (7 pts) Consider the classes shown in the box on the right:

```
public class Employee
{
   public int getPay()
   {
      return 5;
   }
   public void display()
   {
      System.out.println(getPay() + " ");
   }
}
public class CEO extends Employee
{
   public int getPay()
   {
      return 100;
   }
   public void bonus()
   {
      System.out.println( (2*getPay()) + " ");
   }
}
```

      i. (5 pts) What is the printed output (if any or if error) at lines 3-5 below:

```
Line0: Employee emp = new Employee();
Line1: Employee ceo = new CEO();
Line2: CEO ceo2 = new CEO();
Line3: ceo.display();
Line4: ceo2.display();
Line5: emp.display();
```

Output:

```
Line3: 100
Line4: 100
Line5: 5
```

      ii. (2 pts) What is the printed output (if any or if error) at the additional lines below:

```
Line6: ceo2.bonus();
Line7: emp.bonus();
```

Output:

```
Line6: 200
Line7:   not legal expression
```