Questions (5 points)

1. (1 point) Given two classes Couple and Person as follows:

```java
class Couple implements Serializable{
    private Person one;
    private Person two;
    public Couple(Person a, Person b) {
        one = a;
        two = b;
    }
}

public class Person implements Serializable {
    private String name;
    private String place;
    private int year;
    public Person(String aName, String aPlace, int aYear) {
        name = aName;
        place = aPlace;
        year = aYear;
    }
}
```

Given an instance of Couple as: `Couple t1 = new Couple(new Person(“Smith”, “London”, 1934), new Person(“Jones”, “Paris”, 1945));` Show the serialized form of the given Couple instance and explanation as demonstrated in Figure 4.9 of the text.

Answer:

<table>
<thead>
<tr>
<th>Class</th>
<th>Field</th>
<th>Value</th>
<th>Type</th>
</tr>
</thead>
<tbody>
<tr>
<td>Couple</td>
<td>8 byte version</td>
<td>h0</td>
<td>class name, version number, handle</td>
</tr>
<tr>
<td>2</td>
<td>Person one, Person two</td>
<td></td>
<td>number, type and name of instance variables</td>
</tr>
<tr>
<td>Person</td>
<td>8 byte version</td>
<td>h1</td>
<td>serialize instance variable one of Couple</td>
</tr>
<tr>
<td>3</td>
<td>int year, java.lang.String name, java.lang.String place</td>
<td></td>
<td>serialize instance variable two of Couple</td>
</tr>
<tr>
<td>1934</td>
<td>5 Smith, 6 London</td>
<td></td>
<td>values of instance variables</td>
</tr>
<tr>
<td>1945</td>
<td>5 Jones, 5 Paris</td>
<td></td>
<td></td>
</tr>
</tbody>
</table>

Explanation:
- **Serialized values**
- **Explanation**
2. (1.5 points) Download files Person.java, Connection.java, Client.java, and TCPServer.java from the course website. Modify Person.java and Connection.java, and Client.java so as to perform the following task: the client requests a connection to the server, the server accepts it; the server creates a Person instance (“Byron”, “Vancouver”, 5000) and sends it to the client, the client reads it and prints it out. Do the experiment; submit the modified code of Person.java, Connection.java, and Client.java, together with a screenshot of the execution of “java Client”. Assumption: Client has all the information about the object it needs to reconstruct; i.e., Person class info in this case.

Answer:
For Person.java:
Add three accessor methods:
public String getName() {
    return name;
}
public String getPlace() {
    return place;
}
public int getYear() {
    return year;
}
For Connection.java:
Replace PrintWriter by ObjectOutputStream;
Create an instance of Person:
    Person thePerson = new Person("Byron", "Vancouver", 5000);
Replace println() by writeObject():
pOut.writeObject(thePerson);
For Client.java:
try {
    Socket s = new Socket("127.0.0.1", 7896);
    InputStream in = s.getInputStream();
    ObjectInputStream objIn = new ObjectInputStream ( in );
    Person thePerson = (Person) objIn.readObject();
    System.out.println( "name: " + thePerson.getName());
    System.out.println( "place: " + thePerson.getPlace());
    System.out.println( "year:  " + thePerson.getYear());
    s.close();
}
3. (1 point) Download “Client.java for Question 3” from the course website and replace the one in Question 2. Use the original Person.java and TCPServer.java but your modified Connection.java, and run the programs as in Question 2. Submit a screenshot of the execution of “java Client”; then relate the output roughly to the serialization format in Figure 4.9 of the text.

**HINT:** In the output, you may find characters such as p (=hex70), q (=hex71), r (=hex72), ..., x (=hex78). They are keywords used in Java serialization, e.g., t stands for a character string. The length fields consist of two bytes each.

**Answer:**
You should read something like this:

`172\φ'0'5'sr'0'6'Person\µ'31'\'30'U'35'T'2'0'3'I'0'4'yearL'0'4'namet'0'1 8'Ljava/lang/String;L'0'5'placeq'0'~'0'1xp'0'0'19'136't'0'5'Byront'0'9'Vancouver 06: the length of class name Person Person: class name is Person µ'31'\'30'U'35'T': 8-byte version number of the class 2: handle for the Person class '0'3': number of instance variables of the Person class is 3 I: the variable type is integer 04: the length of the variable name year: the variable name L: the variable type is object 04: the length of the variable name name: the variable name t: keyword for the String '0'18'L: String length is 18 L: keywords for the object java/lang/String: class name of the object L: the variable type is object 05: the length of the variable name place: variable name q: the keyword for the reference '0'~'0'1': reference to java/lang/String x: end of the class information '0'0'19'136': value for the year variable. An integer is 4-byte long, so '0' '0' '19' '136' = 0 * 2^24 + 0 * 2^16 + 19 * 2^8 + 136 = 5000 t: keywords for String 05: length of “Byron” “Byron”: value for the name variable t: keyword for String 09: length of “Vancouver”. Vancouver: value for the place variable
4. (1.5 points) A client makes remote procedure calls to a server. The client takes 5 milliseconds to compute the arguments for each request, and the server takes 10 milliseconds to process each request. The local operating system processing time for each send or receive operation is 0.5 milliseconds, and the network time to transmit each request or reply message is 3 milliseconds. Marshalling or unmarshalling takes 0.5 milliseconds per message.

Calculate the time taken by the client to generate and return from two requests:
(i) if it is single-threaded, and
(ii) if it has two threads that can make requests concurrently on a single processor.

Note: You can ignore context-switching times.

Answer:
(i) time per call = calc. args + marshal args + OS send time + message transmission + OS receive time + unmarshall args + execute server procedure + marshall results + OS send time + message transmission + OS receive time + unmarshal args
= 5 + 4*marshal/unmarshal + 4*OS send/receive + 2*message transmission + execute server procedure
= 5 + 4*0.5 + 4*0.5 + 2*3 + 10 ms = 5 + 2 + 2 + 6 + 10 = 25 ms.

Time for two calls = 50 ms.

(ii) threaded calls:
client does calc. args + marshal args + OS send time (call 1) = 5 + .5 = .5 = 6
then calc args + marshal args + OS send time (call 2) = 6
= 12 ms then waits for reply from first call

server gets first call after message transmission + OS receive time + unmarshall args = 6 + 3 + .5 + .5
= 10 ms, takes 10 + 1 to execute, marshal, send at 21 ms
server receives 2nd call before this, but works on it after 21 ms taking 10 + 1, sends it at 32 ms from start
client receives it 3 + 1 = 4 ms later i.e. at 36 ms
(message transmission + OS receive time + unmarshall args) later

Time for 2 calls = 36 ms.