CSC207 Practice Exam Questions

Last Modified: `Monday 26 November 2018`

The exam will test a subset of the topics discussed in this practice exam. It will have fewer questions with a different format.

Solutions will not be posted to these questions. You can post possible solutions and discuss them on the message forum or in any office hours for this course, including:

Date	**Time**	**Location**	**Instructor**
Monday Dec 10	12-2 pm	BA 024	Lindsey
Tuesday Dec 11	5:30-7:30 pm	BA 024	David
Wednesday Dec 12	2-4 pm	BA 025	Paul
Thursday Dec 13	1-3 pm BA 025	Lindsey	

Questions:

1. What are the main components of a class in Java? What are the standard accessibility modifiers for each (e.g., protected, private, etc.)? When would you want to use a non-standard accessibility modifier for a variable? for a method? for a constructor?

2. In what ways are methods similar to constructors? In what ways are they different? (Consider: the syntax for coding each, how they show up in the Java memory model, features such as calling other constructors, whether or not Java provides them by default, returning values, the applicability of terms like "instance" and "static", etc..)

3. Is it possible for a class to have more than one parent class? More than one child class? To implement more than one interface?

4. In which ways are abstract classes and interfaces similar? In what ways are they different? List the conditions which are necessary and sufficient for a: (a) class to be declared abstract and (b) for a method to be abstract.

5. What do the following keywords each mean when used in front of a method: `final`, `static`, `abstract`? Is it possible to have an abstract constructor?

6. List all of the primitive types. How does the memory model reflect the differences between primitive types and objects. Which boolean operator checks to see if two primitive variables have the same value? Which boolean operator checks if two instances of an object are actually the same instance? How can you check if two different instances store the same values? Are all non-primitive types subclasses of the `Object` class? What feature(s) of class `Object` did we override most frequently

in the lectures?

7. What do we mean by “casting”, “autoboxing”, and “wrapper class”. Compare and contrast these terms.

8. Give four examples of subclasses of `Collection` in Java. When would you use each? For example, when would you use an `ArrayList`? How is a subclasses `Collection` similar to an `Array`? How is it different? Is it possible to define a non-generic subclass of `Collection`? How or why not?

9. Write your own generic class and also write a second class with a main method in which you instantiate the generic class.

10. How did we use instances of each of the following classes in the code from Week 8: `Logger`, `Handler`, `Scanner`, `FileInputStream`, `BufferedInputStream`, `ObjectInputStream`, `FileOutputStream`, `BufferedOutputStream`, `ObjectOutputStream`. What is `Serializable` and how can we use it to store information about instances of class `Student`?

11. For each of the following write code so that:

    (a) The main method compiles but does not run to completion.
    (b) The main method compiles and runs, even though one of the methods throws an exception.
    (c) The main method compiles and runs, but prints the call stack trace to the screen twice, at different parts of the program. In other words, the two traces should describe different points in the code.
    (d) The main method compiles and runs, even though an exception is thrown from inside a call block.
    (e) The main method compiles but does not run. However, between the moment when the last exception is thrown and the end of execution, the message “This is a message.” appears on the screen.

12. We discussed the following design patterns in class: Iterator, Observer, Strategy, MVC, Dependency Injection, and Factory Method. When would you want to use each pattern? Describe a situation where the Iterator pattern would be useful. Do that again for each of the other patterns. Describe an alternative to the Observer pattern, the Iterator pattern, and to the Strategy pattern. Which is better design according to SOLID, your alternative, or the design patterns? Explain your answer for each pattern.

13. Modify the file `SupSubDemo.java` from the Quercus “Files” –> “Week 11” files to demonstrate the lookup rules for Java. In other words, create modified versions of the file to check when Java shadows and overrides static/instance methods and static/instance variables.

14. Try the questions in the `regex_practice.txt`. You will be able to find it under the Week 9 Readings on the course website. Also try at least one Regular Expressions Crossword from `regexcrossword.com`.

15. What is a floating point variable? How does the computer store them?

16. Complete an alternative set of CRC cards for the TicketVendor activity that we did during Week 6 lectures. Use as many design patterns as is appropriate. Looking at your cards, is it possible to deduce where each class is instantiated? As the user, where is the entry point into your program? If you move the main method, how does that impact your design?

17. Pretend that you are trying to explain JUnit to someone who knows nothing about it. What is an assertion? What does it mean to pass a test? What is the difference between a fail and an error? What is a "unit test"? What are the three steps in a test? How do you select test cases? Give examples to illustrate your explanations.

18. During Assignment 2, you were given a test file and asked to write code that passes those tests. This is called test-driven development. Give three ways that you might benefit from test-driven development when working at a company where you are part of a team, but non-team members might work on your code in the future.

18. How do you represent the following properties in UML diagrams: static, final, abstract private, public, protected, interfaces. At which stage(s) of development would you use a UML diagram? Compare and contrast UML diagrams, inheritance diagrams, and CRC cards.

19. When is casting useful? Why should we avoid it in our finished product?

20. Think about the process of developing your the design for your Project. What went well? What will you do differently the next time you work on such a large program?

21. What are the SOLID principles of Object Oriented Design? State and explain each.

22. Use each of the SOLID principles to find ways of improving the following code.

Supplementary Code

```
public class Ticket {

  private String event;
  private String buyer;
  private boolean isForSale;
  private static int numSales;

  public Ticket(String event, String buyer){
    this.event = event;
    this.buyer = buyer;
    isForSale = false;
    numSales++;
  }

  public void returnTicket(){
    buyer = "";
    isForSale = true;
  }

  public void sellTicket(String buyer){
    this.buyer = buyer;
    isForSale = false;
    numSales++;
  }

  public String toString(){
    return "this ticket for " + event + " belongs to " + buyer;
  }

  public int getNumSales(){
    return numSales;
  }
}
```

```
public class TrainTicket extends Ticket {

  private String fromCity;
  private String toCity;

  public TrainTicket(String fromCity, String toCity, String buyer){
    super("train ride", buyer);
    this.fromCity = fromCity;
    this.toCity = toCity;
  }

  public void returnTicket(){
    System.out.println("This ticket is for sale again");
  }

  public String getToCity(){
    return toCity;
  }

  public void setToCity(String toCity){
    this.toCity = toCity;
  }
}
```

```
public class TwoWayTrip {

  private TrainTicket departTicket;
  private TrainTicket returnTicket;
  private String startDate;
  private String endDate;

  public TwoWayTrip(String startDate, String endDate, String fromCity, String toCity, St
    departTicket = new TrainTicket(fromCity, toCity, buyer);
    returnTicket = new TrainTicket(toCity, fromCity, buyer);
    this.startDate = startDate;
    this.endDate = endDate;
  }

  public void printItinerary(){
    System.out.println("Start date: " + startDate + ", End date: " + endDate);
  }
}
```

```
public interface SellableAndMoveable {

  public String getPrice();

  public void setPrice(String s);

  public String getOriginalPosition();

  public String getDestination();

}
```