

CSC207 Practice Exam Questions:

Last Modified: Friday 24 November 2017

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
Wed 22 Nov	12:30 – 1:30 PM	BA 2230	Paul
Wed 22 Nov	3–4 PM	BA 2230	Paul
Mon 27 Nov	12:30 – 2:30 PM	BA 3219	Lindsey
Wed 29 Nov	? – ? PM	BA ?	Paul
Wed 29 Nov	? – ?	BA ?	Paul
Mon 4 Dec	12:30 – 2:30 PM	BA 3219	Lindsey
Mon 4 Dec	12:30 – 2:30 PM	BA 3219	Lindsey
Wed 6 Dec	12:30 – 1:30 PM	BA 2230	Paul
Wed 6 Dec	3–4 PM	BA 2230	Paul
Thurs 7 Dec	12:30 – 2:30 PM	BA 3219	Lindsey
Fri 8 Dec	? – ? PM	?	Paul

Questions:

1. What are the main parts of a class in Java? (e.g., methods, etc.) 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`.
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”.
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 one.
10. What is `Serializable`? Why would you want to implement it?
11. (a) Review the “How to Javadoc” slides at:
<http://www.teach.cs.toronto.edu/~csc207h/winter/lectures/javadoc.pdf>.

(b) Find code on your computer that has Javadoc (for example, your A2 code). Navigate to the folder containing the package using the command line (`cd` for “change directory” etc.). Enter the command `javadoc packageName/FileWithMainMethod.java`. What happens to the number of files inside that package? View any html files that are generated, using a web browser.
12. On the course website, in the Week 5 “Exceptions code” you can find: `DemoCheckedAndUnchecked.java`, `UnexpectedNegativeException.java`, and `UnexpectedOddException.java`. Modify the first file to accomplish each of the following results, separately. It is not possible to accomplish them simultaneously.
 - (a) The file compiles but does not run to completion.
 - (b) The file compiles and runs, even though one of the methods throws an exception.
 - (c) The file 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 file compiles and runs, even though an exception is thrown from inside a `try` block.

- (e) The file 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.
13. 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? Code that could benefit from a particular design pattern is often called an “anti-pattern”. Describe the anti-pattern for Iterator. Do that again for each of the other patterns.
 14. Modify the file `SupSubDemo.java` from Week 1 on the website, so that it demonstrates all of the lookup rules for Java. In other words, create modified versions of the file to check when Java shadows or overrides for static/instance methods and static/instance variables.
 15. Try the questions in the `regex_practice.pdf` file on the website Readings. Also try at least one Regular Expressions Crossword from `regexcrossword.com`.
 16. Complete an alternative set of CRC cards for the TicketVendor activity that we did during lecture. 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. 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, String buyer){
        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();  
}
```