# CSC207 Practice Exam Questions

Last Modified: `Sunday 12 August 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. Instead, we will be discussing the answers during the last lecture of the semester. You can also discuss the solutions to these questions on the message forum or in any office hours for this course, including:

Monday 13 August at 4-6pm – David (location BA 7172)
Tuesday 14 August at 5:30 - 6:30 pm – Lindsey (location BA 3201)
Thursday 16 August at 6-7 pm – Lindsey (location BA 5256)
Monday 20 August at 3-5 pm – Lindsey (location BA 5256)

Questions:

1. In order to run a java program, the computer must first compile the `.java` files to create `.class` files. Then the Java Virtual Machine (JVM) runs the `.class` files. After compilation, when the JVM first starts to run the program, what is created in memory first?
   When do instances of an object get created?
   How long do they stay in memory?
   Under what circumstances will the JVM delete an object from memory?

2. 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?

3. 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..)

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

5. 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.

6. What do the following keywords mean when used in front of a method: `final`, `static`, `abstract`.

7. List all of the primitive types.
   How does the memory model reflect the differences between primitive types and objects?
   When are two primitive variables "equal"? When are two objects "equal"?
   What are the differences between `==` and the `equals` method?
   Are all non-primitive types subclasses of the `Object` class? What features of class `Object` did we use most frequently in the lectures, besides the `equals` method?

8. What do we mean by "casting", "autoboxing", and "wrapper class". Compare and contrast these terms.

9. Give four examples of subclasses of `Collection` in Java. Describe a different circumstance for each in which you would require the features of that particular type of collection. For example, how and when would you use an `ArrayList`? How are collections similar to arrays? How are they different? Is it possible to define a non-generic subclass of `Collection`? Why or why not?

10. Write your own generic class and also write a second class in which you instantiate the generic one. What does "instantiate mean"?

11. 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`?

12. **(Optional)** Can you read Javadoc to figure out how to create a main method using classes you have never seen before? In this exercise, you will use the Oracle website to look up the following classes: `JFrame`, `JPanel`, `BoxLayout`, `JCheckBox`, and `ActionListener`. The information on the website is generated from the Javadoc for each class. Use it to write code that displays a JFrame with a panel of checkboxes on it and another panel with an exit button, so that clicking the exit button closes the JFrame.

13. In Week 5, you learned about Exceptions. Write a `main` method that tries to call a different method that can throw an Exception. The method should be located in a different class from `main`. Can you get your `main` method to:

    (a) compile but not run to completion.
    (b) compile and run, even though an exception is thrown.
    (c) compile and run, but print the 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) compile and run, even though an exception is thrown from inside a `catch` 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.

14. 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 solution to the Observer pattern, the Iterator pattern, and to the Strategy pattern.

15. Demonstrate the lookup rules for Java by creating a parent class and child class that each contain a static variable with the same name, instance variable with the same, static method with the same signature and instance method with the same signature. Let the parent class be called `Parent` and the child class be called `Child`. Include the following lines in your main method:
    ```
    Parent var1 = new Parent();
    Parent var2 = new Child();
    Child var3 = new Child();
    ```
    Then try printing the value of each variable and the return value of each method for `var1`, `var2`, and `var3`.
    When does Java shadows (use the code from the parent class) and when does it override (use the code from the child class?

16. Try the questions in the "`regex_practice`" file, on the website under Week 9. **Also try at least one Regular Expressions Crossword from** `regexcrossword.com`.

17. What is a floating point variable? What examples of floating point issues have we seen?

18. 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?

18. 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"?

19. What is version control? What do the following commands do?

```
git pull
git add
git commit
git push
```

    Is this the correct order in which to use these commands? Are there circumstances when you would use them in a different order?

20. Take a look at the UML diagram for your Phase 1. How much of that diagram has changed? What major changes occurred between Phase 1 and Phase 2? For each major change, why is the new design better than the previous? Did the change involve implementing a design pattern? Does the new design follow any of the SOLID principles better than your Phase 1 design?

21. Consider the code that starts on the next page. For each SOLID principle, ask the question:

    Does the code violate this principle?
    If so, on which line(s)?
    How can I fix the code so that it better implements this principle?

22. Consider the code that starts on the next page. Are there any design patterns that can be used to improve or extend it?

23. In the code that starts on the next page, figure out what (if anything) is inherited by each class.

24. How would you recognize each of the following, when looking at someone else's code:
    (a) Cyclomatic complexity
    (b) Data clump
    (c) Excessively long identifiers
    (d) Duplicated code
    (e) Shotgun surgery

```
public class Ticket {

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

  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)
  }
}
```