CISC 3120 Final Review Guide

Main Topics

Java Fundamentals

I expect you to have a basic mastery of control structures (loops, conditionals), statements, arrays, function declaration, definition, and invocation, and language syntax. I won't be testing you explicitly on this, but of course you won't be able to write much code without this! Exam questions will focus on:

Basic object-oriented syntax and terminology (class, object, instance variable, subclass, etc)
Primitive vs reference types.
Java API, packages.
Public, private, and package access.
Polymorphism: inheritance and interfaces
Overloading and overriding
Object lifecycle: object construction, garbage collection.
Static versus instance members
Design patterns in Java, especially Decorator, Strategy, MVC.

Professional Practice

Style and Documentation
Version control and Git/GitHub

Graphics and User Interface

Basics of Swing Basics of Swing application construction
Drawing with Graphics objects
UI components and events: interfaces, event handlers

Network Programming

Streams; Exceptions.
Protocols, IP, TCP, and Sockets
Websockets

Structure and Tactics

See the midterm review sheet!

The last page of this document will be appended to the exam, a very condensed API! That is, I'll give you this sheet of method names and parameters; you will need to know how/where to use them. (Of course, not all of these will necessarily appear on the exam.) If you need more specialized methods, I will describe them in the exam question.

1. Here are two of the CheckStrategy examples from the reading

```
public class StartWithT implements CheckStrategy
{     public boolean check(String s)
      {     if( s == null || s.length() == 0) return false;
            return s.charAt(0) == 't';
      }
}


public class LongerThanN implements CheckStrategy
{     public LongerThanN(size n) {this.n = n;}
      private int n;

      public boolean check(String s)
      {     if(s == null) return false;
            return s.length() > n;
      }
}
```

Write a `Strategy` class that checks to see if a word starts with a given character—just as `LongerThanN` checks for words longer than a given length.

2. You are implementing an endless runner. In your initial release, the character will be able to walk or run, but in future releases, you may be adding flying, swimming, crawling underground, and so on. Design a Strategy (an interface) to help you deal with this uncertain range of possibilities. Give examples (names) of classes that might implement this strategy.

3. The reading claims that the Java API classes `Reader` and `BufferedReader` are a good example of the Decorator pattern. What is the `Reader` class?

a) A class that reads input from a file or the keyboard.
b) An abstract class with methods for reading characters from some source.
c) A class that implements the `Readable` and `Closeable` interfaces.
d) b and c.
e) All of the above.

4. As far as you can tell from the API+documentation, what is the object-oriented relationship between `BufferedReader` and `Reader`?

a) `BufferedReader` IS-A `Reader`
b) `BufferedReader` HAS-A `Reader`
c) Both a and b
d) Neither a nor b

5. So how does a `BufferedReader` "decorate" a `Reader` object (that is, an object of some concrete subclass of `Reader`)?

a) It separates (1) the user's request to read something from (2) the action of reading directly from the character source.
b) It "collects" a bunch of read requests, then issues them all at once to the character source.
c) It reads a big chunk of characters from the character source, then uses that chunk to respond to read requests.
d) It counts the number of lines read by the `Reader`.

6. Which of these other descendants of `Reader` are Decorators? (Possible multiple answers!)

a) CharArrayReader
b) FilterReader
c) InputStreamReader
d) LineNumberReader
e) StringReader

7. Which sentence from the reading is the best capsule explanation for the Decorator pattern?

a) To the client the decorator is invisible.
b) You can think of a decorator as a shell around the object decorated.
c) Decorators are used to provide additional functionality to an object of some kind.
d) The key to a successful Decorator is to have a set of objects defined by an interface.
e) The decorator implements the same interface as the object it decorates.

8. I'm working on a graphic animation library. One fundamental element is a `Shape`, which has the behaviors

```
void paint(Graphics g)
Point getCenter()
void move(int deltaX, int deltaY)
```

I would also like some `Shapes` to move automatically (say, some number of pixels per frame), some `Shapes` to rotate automatically (some number of degrees per frame), and some `Shapes` to shrink automatically (some number of pixels per frame). And, of course, some `Shapes` may need to do several of these things at once.

Come up with a Decorator-based design for this problem.

9. The Iterator design pattern is most closely related to which part of Java?

a) Arrays
b) Enums
c) The enhanced for loop
d) Primitive type wrapper classes

10. Which of these is not a good reason to implement an Iterator?

a) If you change your mind about how you store your data, you don't want to change how you traverse it.
b) You want to be sure that you can "visit" all elements in the most efficient way possible.
c) You want to be able to traverse your complex data in several different ways.
d) You don't want users to have to know the details of how you store data.

11. What is the essential purpose of the Model-View-Controller pattern?

a) Make it easier to "plug in" different GUI components
b) Make it easier to combine GUIs and networks (such as the WWW)
c) Separate information from how it is presented
d) Separate the GUI into 'display' and 'control' elements
e) b and c

12. Which of these are typical relationships among the Model, View, and Controller?

a) Controller tells Model to update its data
b) Model tells View that data has changed.
c) Model sends an "update" event to the Controller
d) a and b
e) a and c

13. Think about the base 4 calculator. Which MVC component(s) does the `Base4CalcState` (the "actual" calculator) represent?

a) Model
b) View
c) Controller
d) a and b
e) b and c

14. In an application using the MVC pattern, how many of each element is advisable?

a) Only one of each element
b) One model; as many views and controllers as necessary
c) One view; as many models and controllers as necessary
d) One controller; as many view and models as necessary
e) As many of each element as necessary

15. True or false: a Swing component can be part of the View, or part of the Controller, but not both.

a) True
b) False

16. Do the exercise described on page 6 of "simple MVC example in Java" from the reading. The Clock code as given is on GitHub at https://github.com/BC-CISC3120-S17/class19-code

The exercise instructions:

- Remove the tick button
- Add a `Timer` object to `ClockController`
  - Remember to `start()` it after creating it
- Create a class to listen to the timer and advance the clock
  - It must implement `ActionListener`
  - `actionPerformed()` must increment the clock every second (1000 milliseconds)
  - You may use an anonymous inner class, similar to those that listen to the buttons
  - You can use the `tickButton` `actionPerformed()` logic
- No changes in `ClockView` or `ClockModel`

Swing allows you to create a `Timer` that will send an event once or repeatedly after a timeout that you set.

The event is an `ActionEvent`. The listener must implement the `ActionListener` interface

How to create a `Timer`: `Timer t = new Timer(intervalInMillisecs, listener);`
How to start and stop a `Timer`: `t.start(); t.stop();`
How to tell a `Timer` to fire only once: `t.setRepeats(false);`

17. Which sort of stream is "lower level," or, closer to the hardware?
    a) `Connection stream`
    b) Chain `stream`

18. Serialization saves not just an object, but the
    a) object chart
    b) object graph
    c) object diagram
    d) subclass tree

19. `Serializable` is a *marker* interface, which means

    a) It contains a single method called `marker()` (the signature may vary across different marker interfaces)
    b) It contains `mark()` and `unmark()` methods, as well as any other methods specific to the interface
    c) It contains no methods
    d) It is in the package `javax.sharpie`.

20. When is a `NotSerializableException` exception thrown (thrown)?

a) When you try to serialize an object of a class that doesn't implement `Serializable`.
b) When you try to serialize an object that contains an object of a class that doesn't implement `Serializable`.
c) When you try to deserialize an object of a class that doesn't implement `Serializable`.
d) a and b
e) a, b, and c

21. What is "risky" about the object serialization process?

a) Creating an `ObjectInputStream` object
b) Creating an `ObjectOutputStream` object
c) Reading an object from an `ObjectInputStream`
d) Writing an object to an `ObjectOutputStream`
e) All of the above

22. Consider the Quiz Card code on p. 457. An exception is dealt with in a `catch` block—a message is written to the console.

(a) What are the other possible places this exception could be handled?
(b) Revise this code so that the exception is handled by communicating with the user through the GUI.

23. Consider the code snippet on p. 463, which shows how the BeatBox program saves a drum "pattern."

Given that these patterns are represented in the GUI by `JCheckBox` objects, it kinda seems like the saving process should involve serializing the 256 `JCheckBox` objects that represent the pattern. But it doesn't. Give at least 2 solid reasons the book uses the approach it does.

24. What is the difference between a `ServerSocket` and a `Socket`?

a) The server's job is to connect; the client's job is to accept.
b) The client's job is to connect; the server's job is to accept.
c) The server produces information; the client consumes it.
d) The client produces information; the client consumes it.

25. How does a `ServerSocket` object let us communicate with the client?

a) It uses classes and methods from Java's `SocketStream` API.
b) Provides an `InputStream` or `OutputStream` object we can use to read/write.
c) Provides a `Socket` object representing a connection with the client.
d) Connects the local keyboard and screen with the client.

26. Suppose I wanted to write a version of the `DailyAdviceClient` for people having especially bad days. I write the following in the `try` block:

```
Socket s = new Socket("127.0.0.1", 4242);
InputStreamReader streamReader = new InputStreamReader(s.getInputStream());
BufferedReader reader = new BufferedReader(streamReader);

String advice = reader.readLine();
System.out.println("Today you should: " + advice);

advice = reader.readLine();
System.out.println("Today you also should: " + advice);

reader.close();
```

What should the result be?

a) Two fresh and delicious pieces of advice.
b) The client application will throw an exception.
c) The server application will throw an exception.
d) Both applications will "freeze."

(What *actually* happens?)

27. Suppose I want the `DailyAdviceServer` to be a little more helpful; when a new connection is made, it should identify itself to the client, so in the `while` loop I write

```
Socket sock = serverSock.accept();
PrintWriter writer = new PrintWriter(sock.getOutputStream());
writer.println("Welcome to the Daily Advice Server!");
String advice = getAdvice();
writer.println(advice);
writer.close();
System.out.println(advice);
```

What will be the result?

a) The client user sees "Welcome to the Daily Advice Server!" followed by a yummy piece of advice.
b) The client user sees some strange advice.
c) The client application throws an exception.
d) The server application throws an exception.
e) Both applications freeze.

28. The `ServerSocket` `accept()` method is blocking. This means

a) All other clients are prevented from accessing the server until it finishes talking with this client.
b) Nothing else is going to happen in the program until the `accept()` method finishes making its connection.
c) The program doesn't need to use the CPU until the `accept()` method hears from a client.
d) Both a and b.
e) Both b and c.

29. What might a “non-blocking” `accept()` method do differently?

a) Communicate with binary data instead of text.
b) Return immediately, even if no client has tried to connect.
c) Refuse clients if the server is too busy.
d) Use a different port number.

Graphics

drawLine(int x1, int y1, int x2, int y2)
drawOval(int x, int y, int width, int height)
drawRect(int x, int y, int width, int height)
drawString(String str, int x, int y)
setColor(Color c)

JTextField/JTextArea

JTextField()
JTextField(int columns)
JTextField(String text)
JTextField(String text, int columns)
String getText()
setText(String t)

Component

addKeyListener(KeyListener l)
addMouseListener(MouseListener l)
addActionListener(ActionListener l)

ActionListener

actionPerformed(ActionEvent e)

KeyListener

keyPressed(KeyEvent e)
keyReleased(KeyEvent e)
keyTyped(KeyEvent e)

MouseListener

mouseClicked(MouseEvent e)
mouseEntered(MouseEvent e)
mouseExited(MouseEvent e)
mousePressed(MouseEvent e)
mouseReleased(MouseEvent e)

EventObject

Object getSource()

Observer

update(Observable obs, Object o)

Observable

addObserver(Observer o)
notifyObservers()
setChanged()

Socket

connect(SocketAddress endpoint)
close()
InetAddress getLocalAddress()
InputStream getInputStream()
OutputStream getOutputStream()

ServerSocket

Socket accept()

ServerEndpoint/ ClientEndPoint

onClose(Session session, CloseReason reason)
onError(Session session, Throwable thr)
onOpen(Session session)
onMessage(String message, Session session)
onMessage(Object object, Session session)

Session

RemoteEndpoint.Basic getBasicRemote()
Map<String,Object> getUserProperties()

Encoder.Text<T>

String encode(T object)

Decoder.Text<T>

T decode(String s)
boolean willDecode(String s)

Map<String,Object>

get(String key)
put(String key, Object value)

RemoteEndpoint.Basic

sendObject(Object data)
sendText(String text)

String

String toUpperCase()
String substring(int begin)
String substring(int begin, int end)
int indexOf(int ch)
int indexOf(String s)

Enum

int compareTo(E o)
String toString()
E valueOf(String)

Object

Class getClass()

JsonObject

add()
build()
getString()

Json

[static] createObjectBuilder()
[static] createReader(Reader r)

(you may detach this page)