Midterm Answers

Note: these answers are based on the 'B' version of the midterm. If you wrote the 'A' version, the same questions appear, but in a different order. In some cases, the wording between exams may be different, so check the wording carefully before reporting errors. Also note: when this .pptx file is run as a presentation, the answers appear after you select the spacebar. It can thus be used as practice for future exams.

Part A: Multiple Choice Questions – worth 1 mark each

Choose the single best answer for each question.
ANSWER ON THE SCANTRON

2. **TRUE** (a) or **FALSE** (b): a base class will have fewer non-private members than its derived class(es)

3. **TRUE** (a) or **FALSE** (b): Whenever any new object is instantiated, `Object`'s no-arg constructor will be loaded first, before any other constructors

4. Which one of the following is ***NOT TRUE*** of constructor chaining?

(a) it leads to *code reuse*, ensuring that code does not need to be repeated in each constructor
(b) it helps to make the client interface more intuitive by presenting a standard set of features to the user of the code
(c) once build, chained constructors generally require less time to maintain and correct
(d) it allows code to be executed faster
(e) none of the above

5. **TRUE** (a) or **FALSE** (b): a static method declared in a superclass will not be visible in its subclasses, even if the method is declared as public.

6. **TRUE** (a) or **FALSE** (b): you cannot chain to a private method

7. Which one of the following is ***NOT TRUE*** of getters?
   (a) they are useful for hiding private data from clients
   (b) they typically return the data type of the private field they 'get' their data from
   (c) along with setters, they are the only kind of method that should talk directly to private fields
   (d) using getters is not considered to be a 'best practice' amongst programmers
   (e) none of the above

8. Which one of the following is used in UML to signal that a class is abstract?
   (a) underlin (b) *talic* (c) **boldface** (d) CAPITALIZE (e) none of the above

9. Which one of the following is ***NOT*** an advantage of using an ArrayList over a regular array?
   (a) ArrayLists are resizable; regular arrays are of a fixed size, set when the array is instantiated.
   (b) ArrayLists are full-blown objects; as such they have many more features than regular arrays.
   (c) Like regular arrays, an ArrayList can be parameterized with both primitive and reference types.
   (d) You can remove elements from the middle of an ArrayList and resize it automatically; this is not possible with regular arrays.
   (e) none of the above

10. Which one of the following symbols is used to indicate that one class inherits from another class (i.e. an *'is a'* relationship involving one object to many others):

(a)  (b)  (c)  (d)  (e) none of the above

11. Which one of the following is the Java header equivalent of the UML statement

```
+add(first: T[], second: T[]): <T> T[]
```

(a) `public static void add(T[] first, T[] second, <T> T[])`
(b) `public static T[] add(<T> T[] first, <T> T[] second)`
(c) `public static <T> T[] add(T[] first, T[] second)`
(d) `public <T> T[] static add(T[] first, T[] second)`
(e) none of the above

12. **TRUE** (a) or **FALSE** (b): if a superclass has any constructors, then, at very least, one of them must be a default constructor, even if that constructor contains no code.

13. Which one of the following words best describes the relationship between the Algonquin College campus and its parking spaces?

(a) association (b) composition (c) aggregation (d) aggression (e) none of the above

14. If a method is declared with a `public` access modifier, then its visibility is limited to

(a) the class (b) the package (c) subclasses
(d) the entire project (e) none of the above

15. Which one of the following *warnings* should you ***NOT*** ignore, if Eclipse flags it in your code?

(a) failure to properly close an object, such as an instantiated Scanner
(b) failure to resolve a raw type warning, for a generically-typed ArrayList
(c) failure to initialize a variable, so that its value is defined at run time
(d) failure to use a class or method that is declared but never used
(e) none of the above, since warnings can always be ignored

16. **TRUE** (a) or **FALSE** (b): abstract classes can't be instantiated, hence they do not have constructors

17. **TRUE** (a) or **FALSE** (b): If a class is abstract, then all of its members must be abstract

18. Which one of the following is ***NOT*** a good reason to use Exceptions (choose the single best answer) [Clarified during the test as: which one of the choices is the poorest choice (as a reason to use Exceptions).]

(a) they can be computationally costly, especially if overused
(b) they separate the source of the exception from the handling of it, making it harder to track down
(c) a single exception handler can capture a wide range of possible subclass exceptions
(d) wrapped too closely around the site of the exception, they may lead to other problems
(e) none of the above

19. **TRUE** (a) or **FALSE** (b): public superclass methods can only be accessed in their subclasses by preceding their name with '`super`.', for example, `super.methodName()`.

20. When `public static void main(String[] args)` is written in a UML class diagram, it will appear as:

(a) `+main(args: String[]): void`
(b) `-main(args: String[]): void`
(c) `+main(args: String[]): void`
(d) `+main(): void`
(e) none of the above

CST8284 – Object-Oriented Programming (Java) – Sec. 300

Hybrid Quiz

21. The passing mechanism Java uses to access objects is most accurately described as:

(a) *pass by reference*—an address is passed which allows the object to be accessed directly
(b) *pass by reference*—a number is passed, which is the offset of the object code in memory
(c) *pass by value*—a number is passed, which is no different than an integer
(d) *pass by value, but with modifications*—a number is passed, but it allows the user to access an address in memory
(e) none of the above

22. **TRUE** (a) or **FALSE** (b): The `File` class contains methods that allow the user to add or delete items stored in a file located on, for example, a hard drive or USB drive.

23. Which *one* of the following is ***NOT*** one of the basic reference types in Java?

(a) a static type (b) a class (c) an interface (d) an array (e) none of the above

24. **TRUE**(a) or **FALSE** (b): The correct way to read all the objects out of the file is to loop through the file using the `readObject()` method repeatedly until the `EOFException` is generated.

25. Which one of the following statements is ***NOT TRUE*** of the regular *for* and the *enhanced for* loop

(a) You use a colon (:) in the enhanced for loop and semi-colons (;) in a regular *for* loop.
(b) You can only get data *out* of an array with an *enhanced for* loop; you cannot set the data into an array (at least not without creating an index variable that acts as it would in a regular *for* loop).
c) Unlike regular *for* loops, you cannot properly access 2-dimensional arrays with *enhanced for* loops
d) You can only count up with an *enhanced for*; with a regular *for* loop you can count up and down.
e) none of the above

26. Setting an object's value to `null` has which one of the following effects?

(a) it causes a `nullPointerException` to be triggered in the code
(b) it loads the default `Null` object
(c) it causes the `final()` method to be executed
(d) it invokes the garbage collector
(e) none of the above

27. When you make a definition like:

```
Scanner scanner = new Scanner(System.in);
```

which one of the following most accurately describes what happens after this line is executed:

(a) `scanner` holds a new `Scanner` object
(b) `scanner` holds an address that points to a new instance of a `Scanner` object
(c) `scanner` holds an ID number that is tied to an address that points to a new instance of a `Scanner` object
(d) `scanner` holds an ID number that is tied to an address that points to the Scanner class initially loaded by the class loader during start-up
(e) none of the above

28. Which one of the following best describes what a stream is in Java?

(a) It is a structure that dynamically allow data to be alternately added or removed from either end of the structure
(b) it is a stack that allows data to be add or removed from one end of the structure only
(c) it is mechanism for transporting data to and from files
(d) it is special type of array that allows data to be inserted at any location in the array
(e) none of the above

29. TRUE (a) or **FALSE** (b): in Java, one only ever passes a value to a method, never an address

Part B: Terminology – worth 1 mark each

FILL IN THE SINGLE BEST ANSWER.

USE ONE WORD ONLY IN EACH SPACE; DO NOT USE ACRONYMS

30. The angular brackets < > are referred to as the _____*diamond*_____ notation.

31. The properties and methods of an object are collectively referred to as its _____*members*_____.

32. In OOP, any class which has two or more parent classes is said to have

_____*multiple*_____ _____*inheritance*_____, (*2 words*) which is not permitted in Java

33. The declaration `<T extends GeometricObject>` is called a _____*bounded*_____ generic type (due to the fact that T is prevented from being just *any* type of object).

34. Inheritance promotes _____*code*_____ _____*reuse*_____ *(2 words*), meaning you build your code only once in the superclass, rather than recreate it again multiple times in subclasses

Part C: Short Written Answers – marks as indicated.

35. Can an abstract method be declared in a final class? Explain why or why not in one or two concise sentences in the space provided below

An abstract method can only be declared in an abstract class, and an abstract class must be extended to a concrete subclass before it can be instantiated. By definition, a final class *cannot* be extended to a subclass, and therefore an abstract method cannot be declared in a final class.

Part D: Spot the error – worth 12 marks total

36. The code shown below on the next three pages is based on the UML diagram (on the next page). A new Building object is instantiated by choosing the first item in the BuildingLauncher class menu. The Building contains 100 apartments, numbered from 100 to 109 on the first floor, 200 to 209 on the second, etc. Each of the 100 Apartment objects is instantiated when the Building is instantiated. Each apartment contains a Kitchen, Bathroom, and Living room, along with 0, 1, or 2 Bedrooms. (The number of Bedrooms is assigned randomly, using the Math.Random() function, which generates a double value greater than or equal to 0, and less than 1.0.)

In addition to instantiating a new Building, the user can get Apartment room size information on any apartment by selecting the second choice form the menu, and then entering the apartment number; or select '3' from the menu to exit. Typical output is shown below:

Read before you proceed

(1) You may assume that the overall construction of the code shown below is essentially correct even if its implementation is unnecessarily bloated and flawed in many critical areas. Your job is not to critique the construction of the program, but simply to find *specific* compile- and run-time/logic errors. Remember: ONLY INDICATE ERRORS THAT ACTUALLY EXIST; DO NOT FLAG AS ERRORS *THINGS THAT YOU DISAGREE WITH*.

(2) If you find an error that seems to occur more than once, chances are that that code is in fact correct, and the real error lies elsewhere at another location, and only once. In other words, you don't get marks for fixing multiple apparent mistakes if a single correction would have fixed all those mistakes at once.

(3) Even if an error *does* occur more than once, you only get credit for the first occurrence of that error, not for repeat errors of the same kind.

(4) Your comments must clearly identify the nature of the error. Comments like "something is wrong here" are not specific enough to warrant marks.

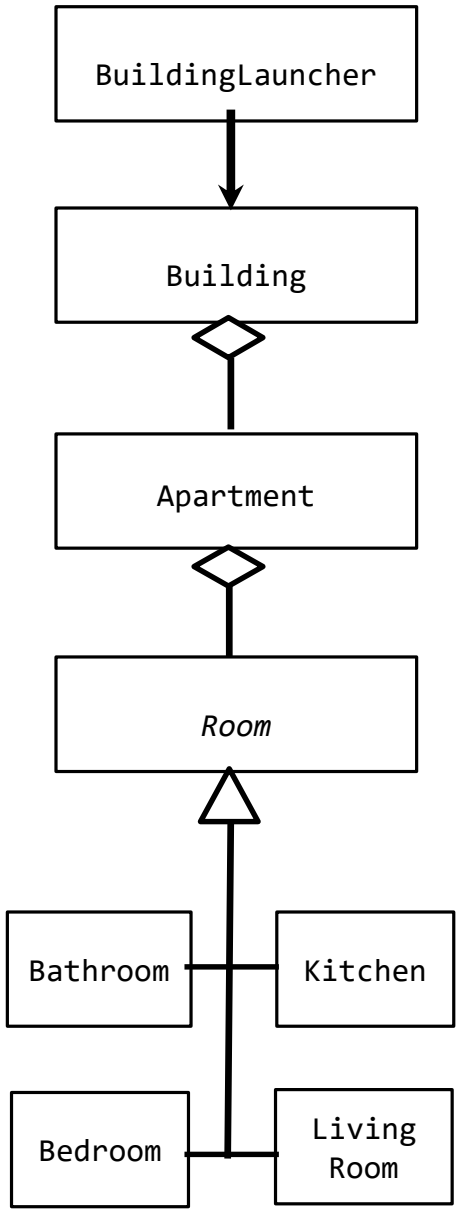
(5) CIRCLE each number and give a brief description of the error itself in the box at right. ***DO NOT* draw lines across the page to connect each error to its description in the box.**

(6) You cannot get additional marks for adding errors beyond the space provided. Therefore, do not initially list as errors things which you suspect are wrong (but don't really know for sure) since you're wasting valuable space with guesses, which probably won't earn you any marks, while taking away from the space that could be used to report legitimate errors. Therefore, list the errors that you are absolutely certain of first, and only list your guesses when you've exhausted all obvious errors.

```
Select one of the following:
1. Load new building
2. Get apartment information
3. Exit
1

Select one of the following:
1. Load new building
2. Get apartment information
3. Exit
2

Enter apartment number: 209
Apartment information:
Bathroom size:10 X 15
Kitchen size:12 X 24
Livingroom size:20 X 15
Bedroom size:12 X 12
Bedroom size:12 X 12
```



```
public class BuildingLauncher {
   public static Building aptBuilding;
   public static int choice = 0;

   public static void main(String[] args) {
            java.util.Scanner scanner = new java.util.Scanner(System.in);
            do {
               boolean badEntry = true;
               do {
                 System.out.println("Select one of the following:\n1. " +
                  "Load new building\n2. Get apartment information\n3. Exit");
                  try {
                          choice = scanner.nextInt();
                          badEntry = false;
                        } catch (InputMismatchException) {
                          System.out.println("Bad value input; please enter a valid number");
                  }
               } while (badEntry);
               switch (choice) {
                  case 1: setAptBuilding(new Building(100)); break;
                  case 2: System.out.println("Enter apartment number :");
                          System.out.println ("Apartment information:\n" +
                            getAptBuilding().getApartments.get(scanner.nextInt()).toString());
                          break;
                  case 3: System.out.println("Exiting");
                  default: System.out.println("Bad value input");
               }
            } while (choice != 3)
   }

   public static void setAptBuilding(Building apartmentBuilding) {
      this.aptBuilding = apartmentBuilding;
   }

   public static Building getAptBuilding() {return aptBuilding;}
}
```

1 2 22 3 4 5 6 7 8 9

```
import java.util.ArrayList;

public class Building {
           private ArrayList<Apartment> apts = new ArrayList<>();
           public Building(int numOfApartments) {
              for (int i = 0; i < numOfAppartments; i++) {
                 int floor = i/10 +1;
                 int room = i%10;
                 int aptNum = floor * 100 + room;
                 getApartments().add(new Apartment(aptNum));
              }
           }

           public ArrayList<Apartment> getApartments(){return apts;}
}
```

10

11

```
import java.util.ArrayList;

public class Apartment {
   private int apartmentNumber;
   public ArrayList<Room> apartmentRooms = new ArrayList<>();
   public Apartment(int aptNum) {
      setApartmentNumber(aptNum);
      getApartmentRooms().add(new Bathroom(10, 15));
      getApartmentRooms().add(new Kitchen(12, 24));
      getApartmentRooms().add(new LivingRoom(20, 15));
      for (int bedrooms; bedrooms < (int)(3*Math.random()); bedrooms++ )
         getApartmentRooms().get(new Bedroom(12, 12));
   }

   public void setApartmentNumber(int aptNum) { apartmentNumber = aptNum; }
   public int getApartmentNumber() {return apartmentNumber;}
   public void setApartmentRooms(ArrayList<Room> aptRms)
      {apartmentRooms = aptRms;}
   public Room getApartmentRooms() {return apartmentRooms;}

   public String toString() {
      String outputString = "";
      for (Room room: getApartmentRooms())
         outputString += room.toString() + "\n";
      return outputString;
   }
}
```

12 13 14 15 16

```
public abstract class Room {
        private int roomWidth;
        private int roomLength;
    17  protected Room(int width, int length){
          setWidth(width); setLength(length);
        }

        public void setWidth(int width) {roomWidth = width;}
        public void setLength(int length) {roomLength = length;}
        public int getWidth() { return roomWidth;}
        public int getLength() { return roomLength;}
        public int getArea() {return getWidth() * getLength();}
        public String toString() {return getWidth() + " X " +
                                                    getLength();}
}
```

18

```
public class Bathroom extend Room {
        Bathroom(int wid, int len) {super(wid, len);}
        public String toString() {return "Bathroom size:" + super.toString();}
}
```

```
public class Bedroom extends Room {
        Bedroom(int wid, int len) {super(wid, len);}
        public String toString() {return "Bedroom size:" + super.toString();}
}
```

```
public class Kitchen extends Room {
        public Kitchen(int wid, int len) {Room(wid, len);}
        public String toString() {return "Kitchen size:" + super.toString();}
}
```

19

20

```
public class Livingroom extends Room {
        Livingroom(int wid, int len) {super(wid, len);}
        public String toString() {return "Livingroom size:" + toString();}
}
```

21

1. Assignment uses single equals ‘=‘, not double equals ‘==‘
2. Did not import exception
3. No guarantee that apartmentBuilding has been set; could be null if the user selects ‘2’ from the menu before selecting ‘1’
4. getApartments is a method; needs ()
5. No guaranteed the number is valid (could be negative, too large, etc.)
6. No attempt to convert apartment number to value stored in arraylist, i.e. Apartment 100 is element 0 in the arraylist
7. Missing break after case ‘3’; code falls to default and prints its message too
8. while() loop is missing semicolon
9. aptBuilding is static, therefore can’t reference this.aptBuilding
10. apts ArrayList should be declared in class, not inside the method, otherwise can’t be seen be getApartments().

11. Extra ‘p’ in numOfApartments, inconsistent with prior use
12. LivingRoom, not Livingroom, i.e. inconsistent with use elsewhere (but see item 20 below)
13. bedrooms not initialized to 0
14. random() used inside for() loop, hence a new random variable is generated each time through the same loop; should be declared as an external variable.
15. Should be using add() to add a new room, not get()
16. Getter for getApartmentRooms should return ArrayList<Rooms>, not Rooms
17. Superclass needs no-arg default constructor
18. extends, not extend
19. Should refer to superclass using super(), not by the superclass name
20. Livingroom is inconsistent with the UML diagram, which says LivingRoom
21. Needs super.toString() to call up to Room’s toString() method from within subclass toString()
22. Need to declare exception variable, e.g. InputMismatchException ex.