CISC275: Midterm #2 Review

1. All possible topics/questions from midterm one are possible topics for midterm 2.
2. Who uses UML diagrams, and why?

UML- Unified Modeling Language

* Consists of nine kinds of modeling diagrams
  + Use case, Class, Sequence, Collaboration, Statechart, Activity, Component, Deployment
* From the POV of the construction trade: Architects design buildings. Builders use the designs to create buildings.
  + Blueprints are the standard graphical language that both architects and builders must learn as part of their trade.
  + UML is the software blueprint language for analysts, designers, and programmers alike to communicate.

1. List and explain the components of a sequence diagram.

Unlike Class and Object diagrams which are static model views, Interaction diagrams are dynamic.

* Describes how objects collaborate.
* Sequence diagram: an interaction diagram that details how operations are carried out – what messages are sent and when
  + Organized according to time, time progresses as you continue down the page
  + Objects involved in operation are listed from left to right according to when they take part in the message sequence.
* Window: the object initiating the sequence of messages.
* Lifeline: vertical dotted line representing the time that an object exists
* Arrow: message call, going from the sender to the top of the activation bar of the message on the receiver’s lifeline.
  + Activation bar: represents the duration of execution of the message.
* Iteration: represented by an asterisk, \*
* Condition: expression in square brackets, []

1. What is a storyboard? Why is it important that it be simply drawn on paper?

Storyboard: a graphic organizer that provides the viewer with a high-level view of a project.

* In Agile software development, a storyboard can help developers quickly get a sense of what work still needs to be completed.
* In Scrum software development, the storyboard may be called the task board.
* As long as the team keeps the storyboard up to date, anyone can see what work has been completed, who’s working on what and what work is left to do.
  + Provides product owner with transparency, also helps the team to visualize the sequence and interconnected of user stories.
  + Should be the first thing clients see if they were to walk into the office, as well as the entire team.

1. What is the difference between overriding and overloading? How do they relate (or not) to object-oriented programming?

**Overriding**: Allows a subclass or child class to provide a specific implementation of a method that is already provided by one of its superclasses or parent classes.

Overriding ---> runtime

• At runtime we know the actual type, not just the reference type.

• Overriding if two methods have same signature.

@Override, Polymorphism, Runtime, Dispatch Vector

**Overloading**: Re-writing the same method multiple times with different parameters, in the same class

• Allows a class to have more than one method of the same name, if argument lists (parameters) are different.

◦ Similar to constructor overloading.

Overloading ---> compile time

• At compile time we ONLY know the reference type.

Overload, Compile Time, NOT Polymorphic, Different Signatures

Polymorphism: take the lowest subclass version of methods

1. Write a short code example to demonstrate overriding.

class Cow {

String noise = "moo";

String speak(){

return noise;

}

void converse(Cow c1){

System.out.println( this.speak() + " to you; " + c1.speak() );

}

}

class Calf extends Cow {

String sound = "meh";

String speak(){

return sound;

}

void converse(Calf c1){

System.out.println( this.speak() + "(Calf) to you; " + c1.speak() );

}

}

class Main {

public static void main(String[] a){

Cow c1 = new Cow();

Calf c3 = new Calf();

Cow c2 = c3;

System.out.println( c1.speak() ); //moo

System.out.println( c2.speak() ); //meh

c1.converse(c3); //moo to you: meh

c3.converse(c1); //meh to you: moo

c1.converse(c2); //moo to you: meh

c2.converse(c3); //meh to you: meh

c3.converse(c3); //meh (Calf) to you: meh

}

}

1. Write a short code example to demonstrate overloading.

class Cow {

String noise = "moo";

String speak(){

return noise;

}

void converse(Cow c1){

System.out.println( this.speak() + " to you; " + c1.speak() );

}

}

class Calf extends Cow {

String sound = "meh";

String speak(){

return sound;

}

void converse(Calf c1){

System.out.println( this.speak() + "(Calf) to you; " + c1.speak() );

}

}

class Main {

public static void main(String[] a){

Cow c1 = new Cow();

Calf c3 = new Calf();

Cow c2 = c3;

System.out.println( c1.speak() ); //moo

System.out.println( c2.speak() ); //meh

c1.converse(c3); //moo to you: meh

c3.converse(c1); //meh to you: moo

c1.converse(c2); //moo to you: meh

c2.converse(c3); //meh to you: meh

c3.converse(c3); //meh (Calf) to you: meh

}

}

\*at compile time, we only know the reference type (in this case ---> c2 is a Cow, not a Calf)

1. What code controls whether members of a class are considered duplicates by a hashed data structure? Write an example with working methods.

hashCode() controls whether members of a class are considered duplicates by a hashed data structure.

* Every object has a hashCode() method associated with it.
  + hashCode() returns an int representing the hash value
* If two objects are equal according to their equals() method, then their hash codes must be the same!
* IMPORTANT!!! ---> hashCode() is always checked first, before equals() method.
* Do NOT write hash functions, Java already writes those for us.

hashCode and equals Java code ex.)

**public** **class** Cat2 {

**public** String name;

**public** Cat2(String n) {

name = n;

}

**public** String toString() {

**return** name;

}

@Override

**public** **boolean** equals(Object o) {

**return** **this**.toString().equalsIgnoreCase(o.toString());

}

@Override

**public** **int** hashCode() {

**return** name.hashCode();

//guaranteed to get a String appropriate hashCode.

}

**public** **static** **void** main(String[] args) {

HashSet<Cat2> cats = **new** HashSet();

cats.add(**new** Cat2("Fido"));

cats.add(**new** Cat2("Phideaux"));

cats.add(**new** Cat2("Fido"));

System.***out***.println(cats);

}

}

1. Explain the order in which equals() and hashCode() are called. Given a series of adds, predict their usage.

hashCode() is called 1st and foremost, THEN if int returned by hashCode match equals() is called.

1. Write the methods hashCode() and equals(), using good software practices.

Scrum software development: