Assignment 5

Eric Wallace

Exercise12.1- It looks about the same.

Exercise12.2- Yes they change with every step to simulate real-life process like migration, overcrowding, breeding, predators, aging etc.

Exercise12.3- It is hard to say based on the animation but, I would assume it follows a general pattern else there would be imbalance.

Exercise12.4- It looks like a pattern of rapid rabbit growth while the fox population drops until they reach a certain point where there is surplus of food the, the fox population grows rapidly and the rabbit drops. From here it cycles over and again.

Exercise12.5- Pretty much the same pattern. Same cycle.

Exercise12.6- If the fox population ever grew too high and ate all the rabbits it could throw this balance off but, I ran a simulation of 104200 steps and neither died off.

Exercise12.7- I feel like a 45 delay is good for me.

Exercise12.8-

Zero: Fox = 214, Rabbit = 782

First step: Fox = 175, Rabbit = 915

Second step: Fox = 188, Rabbit = 1027

Step 4002: Fox = 354, Rabbit = 3672

Step 4003: Fox = 367, Rabbit = 3651

Step 4004: Fox = 385, Rabbit = 3582

Exercise12.9- Same results

Exercise12.10- It breaks it.

Exercise12.11- It seems inherently wrong to not include genders to properly model breeding. The way the program is now if there were only 1 rabbit left it could still breed without a mate.

Exercise12.12- Yes, the program currently allows every rabbit to age up to 40. A litter cannot be greater than 4. The breeding should include other factors like gender and location. The rabbit and fox move at the same rate. All of them could impact the outcome and impact the accuracy of the simulation. I would think if one were to drill down in and be more detailed it could be but, for the purposes here its fine.

Exercise12.13- With a higher rate they produce exponentially then balance into the normal loop with abundance of food. With it lowered the fox die off and the simulation ends with 1 fox because the rabbits didn't produce enough.

Exercise12.14- The fox only has one gender as well. The age again is set. Breeding age is set, litter size all set as constant. Same as before, may not be perfectly accurate but, for the purpose of use its fine.

Exercise12.15- Seems to fall into the normal loop simulation. Didn't fall to zero in a long simulation.

Exercise12.16- They can with certainty if you change the breeding probability to a low enough number for the rabbits the fox will die off of starvation. Setting the breeding age too high can have the same effect of but being eaten before reaching breeding age. There are many ways to cause the population to fade and also several stable versions like the one we began with.

Exercise12.17- Small field size causes quick death with higher chance of survival on a large field.

Exercise12.18- Basically the same. The larger field has larger population due to more space for breeding and less overcrowding.

Exercise12.19- With a field split too small the species will die due to overcrowding so yes it matters how it gets split but, the simulation will work fine as long as one of the fields is not too small.

Exercise12.20- The rabbits die off with the foxes being able to eat them quicker and decreasing starvation.

Exercise12.21- It balances out because even though the fox will consume more around them, starvation will increase across the fox to balance out.

Exercise12.22- Rabbit population depends on all characteristics. If the characteristics are decreased too low the species will die out. Increasing them will find balance no matter the increase. If there are only rabbits, they would fill up the space and only die of age with no fox to eat them. If it were only fox, they would die out from age and starvation. Any imbalance shifted greatly toward the fox would cause an outnumbering of the rabbits which could cause them to collapse if they eat up the rabbits.

Exercise12.23- I did not get a catastrophic outcome with multiple resets and long simulations.

Exercise12.24- The initial population size only impacts for the first few steps, once the foxes reach age it will balance out.

Exercise12.25- Got it.

Exercise12.26- It would probably not be any better. Performance would drop having to loop through the whole field to generate the lists. Could not use lists and just loop the whole field.

Exercise12.27- It says there are animals in the field that are not on the lists. No, dead animals are supposed to be removed.

Exercise12.28-

Similarities:

Instance fields- age, alive, location, field.

Static fields- breeding age, max age, breeding probability, max litter size and rand.

Methods- isAlive(), getLocation(), setLocation(), incrementAge(), giveBirth(), breed(), canBreed(), setDead()

constructors: public Fox(boolean randomAge, Field field, Location location)

The Rabbit class only has one method named run.

The Fox class specific:

instance- foodLevel

static- rabbit food value

methods- hunt(), findFood(), incrementHunger().

Exercise12.29- isAlive, setDead, getLocation, setLocation are all identical.

Exercise12.30- I don't think it really makes a difference.

Exercise12.31- Could just run the simulation to see if it behaves as expected. Should test every change made though. Something like checking whether a fox's hunt method will cause the rabbit to become dead.

Exercise12.32- I made the changes but have errors.

Exercise12.33- It cuts down on redundant code. A cleaner to read file.

Exercise12.34- You need to use Animal instances and call its methods.

Exercise12.35- It is necessary to be abstract if it has abstract methods.

Exercise12.36- It is possible to be abstract even if there are no abstract methods. It can't be instantiated but it can be subclassed.

Exercise12.37- It could.

Exercise12.38- If you click on them it will show if they are declared abstract. If a class has the keyword abstract before the class keyword it is an easy way to identify. AbstractList is extended by concrete classes ArrayList and Vector. AbstractMap is extended by ConcurrentHashMap, ConcurrentSkipListMap, EnumMap, HashMap, IdentityHashMap, TreeMap, and WeakHashMap. AbstractQueue is extended by ArrayBlcokingQueue, ConcurrentLinkedQueue, DelayQueue, LinkeBlockDeque, LinkedBlockingQueue, LinkedTransferQueue, PriorityBlockingQueue, PriorityQueue and SynchronousQueue. AbstractSequentialList is extended by LinkedList. AbstractSet is extended by ConcurrentSkipListSet, CopyOnWriteArraySet, EnumSet, HashSet and TreeSet.

Exercise12.39- You can tell which are abstract. The documentation has a section abstract methods that only shows abstract methods, or just look for the abstract modifier in the method declaration.

Exercise12.40-

Exercise12.41- Did it.

Exercise12.42- The graph shows steps over time. The two curves represent fox and rabbit populations. As time presses on, the rabbit population grows as does the fox until a certain point then, drops and cycles over again.

Exercise12.43- Smaller fields create more unpredictable fluctuations in the patterns. Otherwise behaves as expected.

Exercise12.44-

Step 0- Fox- 175, Rabbit- 915

Step 1- Fox- 188, Rabbit- 1027

Step 2- Fox- 183, Rabbit- 1126

Step 4- Fox- 194, Rabbit- 1309

Step 8- Fox- 230, Rabbit- 1849

Step 16- Fox- 261, Rabbit- 3128

Exercise12.45- Done, appears to work as expected.

Exercise12.46- Did it.

Exercise12.47- Did it.

Exercise12.48- Yes, I moved it over.

Exercise12.49- Not really

Exercise12.50- It is possible to make the changes without impacting other classes. It was a good example of decoupling and encapsulation.

Exercise12.51- I added a MountainLion class which works but with my settings die extremely quickly. I tweaked them to try and find a balance but moved on without finding it.

Exercise12.52- It can be done. I made the changes.

Exercise12.53- No it doesn't. I have errors to fix.

Exercise12.54- No, I have errors.

Exercise12.55- Class fields with public visibility.

Exercise12.56- An interface is supposed to only have abstract public methods and can only be public, static or final.

Exercise12.57- Added the hunter class. I had to go back through and fix a bunch of errors that compiled from about 3 or 4 exercises but it works again. The hunters do not disappear but the fox and mountain lion do every time.

Exercise12.58- The methods that ArrayList has that List does not are ensureCapacity(int minCapacity) and trimToSize(); the methods that LinkedList has that List doesn't are offer(E e), offerFirst(E e), offerLast(E e), peek(), peekFirst(), peekLast(), poll(), pollFirst(), pollLast(), pop(), push(E e).

Exercise12.59- Did it.

Exercise12.60- The sort method mentions interfaces List and Comparator. List methods that have default implementations include replaceAll, sort, spliterator().

Exercise12.61- Did it.

Exercise12.62- In the Simulator class there is the SimulatorView objects. That field uses the interface to enable different kinds of views to be stored in the list. In the constructor a variable is declared, to store different views in the one variable. A GridView object is then assigned and added to the list. The setColor method is called as well. In the simulation method, the list of views is called to get the initial GridView and then checks whether to continue calling the isViable method. The reset method iterates through the list and calls on each of the view's reset method. The updateViews method iterates through the list and calls on the showStatus method which updates the display value.

Exercise12.63- Did it.

Exercise12.64- I had all three up at one time. All worked as expected.

Exercise12.65- Time increments in discrete values in time-based simulations, while event-driven time is incremented differently each time an event happens. Event-driven can run more complex situations efficiently.

Exercise12.66- Practically any work. LinkedList or Hashtable or HashMap will all work. PriorityQueue is the best match though.

Exercise12.68- Yes, it can have normal methods and methods with only the header. A concrete class can't have an abstract method. It has to be declared as abstract. A class can be declared abstract without abstract methods.

Exercise12.69- z, b, u, and u all either implement or extend u, x, G, and x respectively. B, G, U, U, and X do not extend or implement U, X, B, Z, and G respectively. Multiple classes implement or extend x so it is an interface. U, Z, and B are classes. Z extends U, U and B implement X, U implements G and G is an interface.

Exercise12.70- See in project files.

Exercise12.71- According to Oracle docs the PrintJobAdapter is an abstract adapter class for receiving print job events. The methods are empty. It exists as a listener object. It could be useful to have them for peripheral devices and implementing there functionality with programs.

Exercise12.72-

public class Person implements Comparable

{

private float height;

private int age;

public Person(float height, int age)

{

this.height = height;

this.age = age;

}

@Override

public String toString()

{

return "Person{" + "height=" + height + ", age=" + age + '}';

}

@Override

public int compareTo(Person person)

{

return age-person.age;

}

}

public class Main

{

public static void main(String[] args)

{

TreeSet<Person> treeSet = new TreeSet<>();

treeSet.add(new Person(180,16));

treeSet.add(new Person(160,12));

treeSet.add(new Person(190,20));

System.out.println(treeSet);

}

Exercise12.73-

public class UnmodifiableList extends AbstractList

{

private final E[] arr;

@Override

public E get(int i)

{

return arr[i];

}

public UnmodifiableList(Collection<? extends E> c)

{

this.arr = (E[]) c.toArray();

}

}

Chapter14

Exercise14.1- Did it.

Exercise14.2- Did it.

Exercise14.3- addDetails() will overwrite existing data, if a user submits empty string it will replace any existing data.

Exercise14.4- TreeMap can't be replaced in this with HashMap. It doesn't have all the functionality needed for the methods.

Exercise14.5- Did it.

Exercise14.6- It would not always be true. If there are duplicate values the equation would be incorrect.

Exercise14.7- I feel that would be very difficult.

Exercise14.8- I get a NullPointerException which is because you can't call remove on a null object.

Exercise14.9- If the program was running for vital infrastructure like power supply plants, banks, or healthcare providers for instance could cause major disruptions and even death.

Exercise14.10- I would say it is unavoidable to have errors at some point. With everything produced by humans there is a degree of error possibility regardless of skill. All we can do is give it our best, double check and have quality control.

Exercise14.12- It should be reported to the user. Depending on the program the options would vary but, one way would be adding a pop up in the GUI that reports the error.

Exercise14.13- Updated addDetails method.

Exercise14.14- They just check to see if an object exists or return it if possible so they don't have a risk for exception.

Exercise14.15- Errors should always be reported and how depends on the program. If its text-based it should print to screen, if it's GUI it should be visible or generate a pop-up to let the user know.

Exercise14.16- No.

Exercise14.17- Using System.out or System.err, write log files, throw an exception, use a GUI message, play error sounds or messages.

Exercise14.18- If something goes wrong the levels help with filtering out to see where the problems are. It makes the log more readable. It can be turned on and off.

Exercise14.19- Yes, one can just print out to console while the other can have designed messages or pop-ups.

Exercise14.20- Did it

Exercise14.21- No because not every search will return matching results.

Exercise14.22- Entering one as null or empty.

Exercise14.23- The only way is to throw an exception.

Exercise14.24- EOFException- unexpectedly reached the end of file or stream. FileNotFoundException- file on the specified path failed to open. InterruptedIOException- I/O operation was interrupted.

Exercise14.25- SecurityException is unchecked, NoSuchMethodException is checked.

Exercise14.26- Did it.

Exercise14.27- Did it.

Exercise14.28- It can be used to replace System.err.

Exercise14.29- Did it.

Exercise14.30- Did it.

Exercise14.31- p is not changed from null in the code so it will cause exception. It will compile but will throw the exception.

Exercise14.32- Did it.

Exercise14.33- The catch blocks are out of order, the RuntimeException should come first.

Exercise14.34- Did it.

Exercise14.35- It should be unchecked. It's only thrown in the runtime with duplicate keys.

Exercise14.36- The assertion statements are checking book size after adding and after removal and checking for key duplicates.

Exercise14.37- testAddition checks if assertions in adding a contact fail, testRemoval checks for removal fail, testChange checks to see if changes to details failed, testForAdditionError is checking that assertion fails when trying to add duplicate contacts

Exercise14.38- Did it. No need to assert consistent size.

Exercise14.39- Did it.

Exercise14.40- Immutability is very important for proper function. If there are multiple similar fields like having two options for phone numbers there can be an exception if the key is set to one over the other.

Exercise14.41- Files have the information whether the file exists, its attributes, last modified time, owner, file permissions, whether a file is any of the following: directory, executable, hidden, readable, writable, regular.

Exercise14.42- Files.isDirectroy(path), Files.isRegularFile(path).

Exercise14.43- Files.probeContentType(Pathpath) can probe the content type of a file.

Exercise14.45- Did it.

Exercise14.46- Did it.

Exercise14.47- Did it.

Exercise14.48- It is used to read user input from the console.

Exercise14.49- nextBigDecimal, nextBigInteger, nextBoolean, nextByte, nextFloat, nextLong, nextShort.

Exercise14.50- its used as a reader to read input from [System.in](http://system.in/) and as a tokenizer to parse line of string into words.

Exercise14.51- It uses Scanner.nextInt() to extract int values.

Exercise14.52- Only the Post class has to be declared Serializable and in NewsFeed we update the method.

Exercise14.53- While reading no value will be found in the field so it will be set to default for its type.