Section 8 Work Sheet

Below are the design patterns we have talked about with each word in chaos. Fix the words and then use them to answer the questions below:

delBiru	MetrFoaohcd yt
t cjbeOFtorcay Ignentinr drptaAe	
creoarotD oyxPr nonltegiS	

```
public class MysteryOtter {

    private static final MysteryOtter instance =
                                      new MysteryOtter();

    //private constructor to avoid client applications to use
constructor
    private MysteryOtter(){}

    public static MysteryOtter getInstance(){
        return instance;
    }
}
```

Q1: What design pattern is this?

```
public CuteCow CreateCuteCow(String variety){
      if (variety.equals("white"))
            return new WhiteCuteCow();
      if (variety.equals("shining"))
            return new ShiningCuteCow();
      if (variety.equals("rainbow"))
            return new RainbowCuteCow ();
      return new EmptyCow();
}
```

Q2: What design pattern is this?

```java
public class MysteryFlamingo {

    private static MysteryFlamingo instance;

    private MysteryFlamingo(){}

    public static MysteryFlamingo getInstance(){
        if(instance == null){
            instance = new MysteryFlamingo();
        }
        return instance;
    }
}
```

Q3: What design pattern is this?

```java
public class MysteryCat {
    private final String cuteness, temper

    private static Map<String, MysteryCat> instance =
                    new HashMap<String, MysteryCat>;

    private MysteryCat(String cuteness, String temper) {
        this.cuteness = cuteness;
        this.temper = temper;
    }

    public static Cat getInstance(String cuteness,
                                    String temper){
        String key = cuteness + temper;
        if(!instance.containsKey(key)){
            instance.put(key,
                        new MysteryCat(cuteness, temper));
        }
        return instance.get(key);
    }

    . . . . . .
}
```

Q4: What design pattern is this?

```
interface IceCreamFactory {
    IceCream getIceCream();
}

class VanillaFactory implements IceCreamFactory {
    public IceCream getIceCream() {
        return new VanillaIceCream();
    }
}

class ChocolateFactory implements IceCreamFactory {
    public IceCream getIceCream() {
        return new ChocolateIceCream();
    }
}

class StrawberryFactory implements IceCreamFactory {
    public IceCream getIceCream() {
        return new StrawberryIceCream();
    }
}
```

Q5: What design pattern is this?

Q6: Try writing a Builder class and a corresponding class

Q7: Which design pattern specifically requires the class be immutable?

Q8: Which design pattern requires that only one object of the class can ever exist at runtime?

Q9: Which design pattern uses an extra class to store initiation values for an object and eliminates the need for multiple constructors in the original class?

Q10: Matt made a program that can beautifully and perfectly visualize a list on his phone (in his dream). Now he wants write a new and better one base on it. Match the following scenarios to the correct structural design patterns:

Matt wants to make sure it only works for premium users:

Matt wants to make it also able to visualize HashMap and TreeSet on his phone:

Matt wants to make it also able to read content out loudly: