Name: Stamati Morellas COM S 362 - S20 Due 1/24/20 at 11:59pm
Section: 2

Homework 1 – Code Smells & Design Principles

For each problem, study the supplied code, and list the code smells you see. For this exercise the code smells should be one of these:

- long method,
- large class,
- duplicate code (aka cut and paste code),
- long parameter list,
- primitive obsession, and
- magic numbers.

For each code smell,

- list the line numbers of the code where the smell is focused,
- identify the design principle(s) that are violated, and
- identify some program change that the smell would complicate.

---

1. Code fragment 1

```
1   public void onTaxAccountNoChanged(String taxAccountNo) {
2       String errorMessage = TaxAccountNoUtil.validateTaxAccountNo(taxAccountNo);
3       if(errorMessage != null) {
4           showTaxAccountNoValidationError(errorMessage);
5       } else {
6           String taxOfficeIdInput = taxAccountNo.substring(0, 2);
7           Integer taxOfficeId = Integer.valueOf(taxOfficeIdInput);
8
9           String taxOfficeName = taxOfficeRepository.findByNumber(taxOfficeId).getName();
10          showTaxOfficeName(taxOfficeName);
11      }
12  }
```

Smells and locations?

1. Primitive obsession – Occurs at line 6-7
2. Magic numbers – Occurs at lines 6-7

Name: Stamati Morellas COM S 362 - S20 Due 1/24/20 at 11:59pm
Section: 2

Design principles?

1. The primitive obsession code smell violates the information hiding design principle. Passing the value of a string integer and trying to convert it to an integer type can cause accuracy errors, so this should be avoided.
2. The magic numbers code smell violates the information hiding design principle. The values of 0 and 2, which determine the substring size, should be declared as static variables with a name to avoid confusion.

Maintenance Consequences?

1. Changing the type of the method parameter to integer could potentially help fix this, but further refactoring would be required, particularly in the `validateTaxAccountNo()` method in order to make adjustments for the type change.
2. Adding static variables for the substring size could potentially result in more confusion when attempting to refactor since two new elements are being introduced.

Name: Stamati Morellas COM S 362 - S20 Due 1/24/20 at 11:59pm
Section: 2

2. Code fragment 2

```
static WarehouseWorker workerLogin() throws Exception {
        System.out.println(":: WORKER LOGIN ::");
        Scanner sc = new Scanner(System.in);
        System.out.println("Enter username:");
        String username = sc.next();
        int input;
        if(Service.userAndPasswords.containsKey(username) == false) {
            System.out.println("Username doesn't exist, try again or sign up");
            System.out.println("(1) Try again");
            System.out.println("(2) Sign Up");
            input = sc.nextInt();
            if(input == 1) {
                Service.loginHelper();
            }else if(input == 2) {
                Service.signup();
            }else {
                throw new Exception("Bad User Input");
            }
        }
        String check = Service.userAndPasswords.get(username);
        while(true) {
            System.out.println("Enter password:");
            String password = sc.next();
            if(check.equals(password)) {
                System.out.println("Welcome Back!");
                break;
            } else {
                System.out.println("Wrong password!");
                System.out.println("(1) Try with different username");
                System.out.println("(2) Retry password");
                input = sc.nextInt();
                if(input == 1) {
                    Service.loginHelper();
                }
            }
        }
        return getWarehouseWorkerByUsername(username);
    }
```

```
39      static Customer customerLogin() throws Exception{
40          //just a placeholder to look like login page
41          System.out.println(":: CUSTOMER LOGIN ::");
42          Scanner sc = new Scanner(System.in);
43          System.out.println("Enter username:");
44          String username = sc.next();
45          int input;
46          if(Service.userAndPasswords.containsKey(username) == false) {
47              System.out.println("Username doesn't exist, try again or sign up");
48              System.out.println("(1) Try again");
49              System.out.println("(2) Sign Up");
50              input = sc.nextInt();
51              if(input == 1) {
52                  Service.loginHelper();
53              }else if(input == 2) {
54                  Service.signup();
55              }else {
56                  throw new Exception("Bad User Input");
57              }
58          }
59          String check = Service.userAndPasswords.get(username);
60          while(true) {
61              System.out.println("Enter password:");
62              String password = sc.next();
63              if(check.equals(password)) {
64                  System.out.println("Welcome Back!");
65                  break;
66              } else {
67                  System.out.println("Wrong password!");
68                  System.out.println("(1) Try with different username");
69                  System.out.println("(2) Retry password");
70                  input = sc.nextInt();
71                  if(input == 1) {
72                      Service.loginHelper();
73                  }
74              }
75          }
76          Customer cust = findCustomerByUsername(username);
77          return cust;
78      }
```

Smells and locations?

1. Duplicate code – lines 20-37; The logic for the `workerLogin()` and `customerLogin()` methods are almost identical.

Design principles?

1. This violates the abstraction principle, since you could instead create a single method to handle login requests on objects with various types.

Maintenance Consequences?

1. Refactoring the code to implement a single abstract method for logging in for customers and workers could result in conflicts related to incorrect privileges of users if done incorrectly.

Name: Stamati Morellas COM S 362 - S20 Due 1/24/20 at 11:59pm
Section: 2

3. Code Fragment

```
public void createBoard() {
        createTile(0,0,1);
        createTile(0,1,1);
        createTile(0,2,4);
        createTile(0,3,2);
        createTile(1,1,2);
        createTile(1,0,1);
        initializeSpaces();
}
```

Smells and locations?

1. Magic numbers – lines 2-7

Design principles?

1. Hard-coding the tile values when creating the board violates the information hiding design principle. It is hard to tell what the parameter values of the `createTile()` method actually represent

Maintenance Consequences?

1. Since modifying the values of the board at a later stage will be difficult due to this smell, we can re-implement the `createBoard()` method to take in parameters to identify a certain configuration of board to create, as opposed to creating the same one every time, however this could make it more difficult since we will need to write more code to behave in accordance with the type of board that is being made.

Name: Stamati Morellas COM S 362 - S20 Due 1/24/20 at 11:59pm
Section: 2

4. Code fragment 4

```
private static Cart checkOut(Cart customersCart) {
        Scanner sc = new Scanner(System.in);
        Random rand = new Random();
        int c = rand.nextInt(3)+2;
        ArrayList<Item> items = new ArrayList<Item>(customersCart.getItems());
        for(Item i : items) {
            System.out.println("your current cart:");
            displayCart(customersCart.getItems());
            System.out.println("would you like to remove an item? (y/n)");
            if(sc.next().equals("y")) {
                printItemsFromList(customersCart.getItems());
                System.out.println("type the number of the item to remove:");
                customersCart.getItems().remove(sc.nextInt()-1);
            }else {
                break;
            }
        }
        items = customersCart.getItems();
        if(items.size() == 0) {
            System.out.println("your cart is empty");
            return new Cart(items);
        }
        int total = 0;
        for(Item i : items) {
            total += i.getPrice();
        }
        System.out.println("your total is: $" + total);
        String s;
        int shipCost = 7;
        System.out.println("Would you like to ship these items?(y/n)");
        String ans = sc.next();
        String[] ch;
        if(ans.equals("y")) {
            System.out.println("Please enter your address:");
            System.out.println("City, State");
            while(true) {
            s = sc.nextLine();
            ch = s.split(", ");
            //System.out.println("length"+ch.length);
            if(ch.length !=2) {
                System.out.println("Please write it in this format. For instance: (Ames, Iowa)")
                continue;
            }
            break;

        }
            System.out.println("Choose your delivery option:");

            if(ch[1].charAt(0) == 'I' || ch[1].charAt(0) == 'M') {
                System.out.println("(1) 1 day shipping - $" + shipCost);
                System.out.println("(2) 3-4 day shipping - $" + c);
            }
```

Name: Stamati Morellas COM S 362 - S20 Due 1/24/20 at 11:59pm
Section: 2

```
53              else {
54                  shipCost = 10;
55                  c+= 2;
56                  System.out.println("(1) 1 day shipping - $" + shipCost);
57                  System.out.println("(2) 3-4 day shipping - $" + c);
58              }
59              int n = sc.nextInt();
60              if(n==1) {
61                  total+=shipCost;
62              }
63              else if(n==2) {
64                  total+=c;
65              }
66              System.out.println("your total is now: $" + total);
67          }
68          System.out.println("please input your card number:");
69          String cardNumber = sc.next();
70          System.out.println("is " + cardNumber + " valid? (y/n)");
71          if(sc.next().equals("n")) {
72              boolean cardNumberValid = false;
73              while(!cardNumberValid) {
74                  System.out.println("your total is: $" + total);
75                  System.out.println("please input your card number:");
76                  String nextCardNumber = sc.next();
77                  System.out.println("is " + nextCardNumber + " valid? (y/n)");
78                  if(sc.next().equals("y")) {
79                      cardNumberValid = true;
80                  }
81              }
82          }
83          Cart cart = new Cart(items);
84          cart.setDiscount(0 - shipCost);
85          System.out.println("would you like to continue shopping? (y/n)");
86          if(sc.next().equals("y")) {
87              Cart deepCart = addToCart();
88              customersCart.getItems().addAll(deepCart.getItems());
89              return customersCart;
90          }else {
91              return customersCart;
92          }
93          return cart;
94      }
95
```

Smells and locations?

1. Long method

Design principles?

1. The long method code smell violates the separation of concerns design principle. This method can be broken down into smaller components. The different components of the `checkOut()` method could be partitioned into smaller methods in order to aid in readability of the code.

Maintenance Consequences?

1. Attempting to refactor the code to break down the method functionality into smaller components could result in values of components not communicating with each other as expected, since a higher importance will be placed on the scope of variables.

Name: Stamati Morellas COM S 362 - S20 Due 1/24/20 at 11:59pm
Section: 2

5. Code fragment 5

```
public class database {
	public static List<student> studentTable;
	public static List<faculty> facultyTable;
	public static List<Group> groupTable;
	public static List<Major> majorTable;
	public static List<Dorm> dormTable;
	public static List<StudentOrg> studentOrgsTable;
	public static List<Lot> parkingLotTable;
	public static List<Employer> employerTable;
	public static DiningCenter seasons;
	public static DiningCenter conversations;
	public static DiningCenter udcc;
	public static DiningCenter windows;
	public static List<Bus> busTable;
	public static List<BusRoute> busRoutetable;
	public static List<Fair> fairTable;
	public static List<IntramuralSport> intramuralSportTable;
	public static List<GymEquipment> gymequip;
	public static List<GymRental> gymrent;
	public Calendar calendar;

	public database() throws FileNotFoundException {
		......
	}

	/**
	 * Adds fair to to table
	 * @param fair
	 */
	public static void addFair(Fair fair){
		.....
	}

	/**
	 * Returns fair based on building and date
	 * @param building
	 * @param date
	 * @return Fair
	 */
	public static Fair getFair(String building, Date date){
		......
	}

	/**
	 * Removes fair from fair table
	 * @param fair
	 */
	public static void removeFair(Fair fair){
		......
	}

	/**
	 * Adds student to table
	 * @param s
	 */
	public static void addStudent(student s) {
		.....
	}

	/**
	 * Adds bus to table
	 * @param bus
	 */
	public static void removeBusRoute(BusRoute route){
		.....
	}

	/**
	 * Gets busroute based on name
	 * @param routeName
	 * @return BusRoute
	 */
	public BusRoute getBusRoute(String routeName){
		.....
	}

	/**
	 * Gets bus based on number
	 * @param busNumber
	 * @return Bus
	 */
	public Bus getBus(int busNumber){
		.....
	}

	/**
	 * Adds lot to table
	 * @param l
	 */
	public static void addLot(Lot l){
		.....
	}

	/**
	 * Finds a lot
	 * @param id
	 * @return Lot
	 */
	public static Lot findLot(String id){
		.....
	}

	/**
	 * Removes lot
	 * @param l
	 */
	public static void removeLot(Lot l){
		.....
	}

	/**
	 * Adds gym equipment
	 * @param addnew
	 * @return boolean
	 */
	public static boolean addGymEquipment(GymEquipment
addnew){
		.....
	}

	/**
```

Name: Stamati Morellas COM S 362 - S20 Due 1/24/20 at 11:59pm
Section: 2

```
 */
public static void addBus(Bus bus){
        .....
}

/**
 * Removes bus from table
 * @param bus
 */
public static void removeBus(Bus bus){
        .....
}
/**
 * Adds bus route
 * @param route
 */
public static void addBusRoute(BusRoute route){
        .....
}
```

```
 * Finds gym equipment and returns it
 * @param idname
 * @return GymEquipment
 */
public static GymEquipment findGymEquipment(String idname){
        .....
}

/**
 * Removes gym equipment
 * @param idname
 * @return
 */
public static boolean removeGymEquipment(String idname){
        .....
}

public String deleteGymItem(String id){
        .....
}
```

Smells and locations?

1. Large class
2. Primitive obsession

Design principles?

1. The large class code smell violates the separation of concerns design principle. This class could be split into different components (objects) so that the relevant methods for each object are put into smaller classes, which can then be used inside of the `database` class.
2. The primitive obsession code smell violates the information hiding design principle. Strings are sometimes used to represent integer values, which could lead to a misinterpretation of information in the return values of certain methods.

Maintenance Consequences?

1. Creating smaller classes for certain objects and methods in this case could potentially cause difficulty tracking values for certain variables within the smaller classes.
2. Instead of using strings to represent IDs passed to the respective methods, a stronger emphasis can be put on the data types of IDs to coincide more with their actual value. This could require refactoring of other methods in order to function correctly.

6. Code fragment 6

```
26
27@         public static void main(String[] args) throws IOException {
28
29              //Calling constructors of variety of Bloom Filters (Data structures
30              BloomDifferential bloom = new BloomDifferential();
31              BloomFilterFNV filter = bloom.createFilter();
32              NaiveDifferential naive = new NaiveDifferential();
33              HashMap<String,String> table = naive.constructHash();
34
35              //Reading in given number of keys, determining if it is present in
36              readInKeys();
37              int i = 0;
38              startTime = System.nanoTime();
39              for(i = 0; i < 3000; i++)
40              {
41                  bloom.retrieveRecord(keysToSearch.get(i));
42              }
43              endTime = System.nanoTime();
44              bloomTime = calculateDuration();
45              System.out.println("Bloom Time: " + bloomTime);
46
47              startTime = System.nanoTime();
48              for(i = 0; i < 3000; i++)
49              {
50                  naive.retrieveRecord(keysToSearch.get(i));
51              }
52              endTime = System.nanoTime();
53              naiveTime = calculateDuration();
54              System.out.println("Naive Time: " + naiveTime);
55              bloom.getAverageTime();
56              naive.getAverageTime();
57          }
58
```

Smells and locations?

1. Magic numbers – lines 39, 48

Design principles?

1. The magic number code smell violates the information hiding principle. Here, we see that during the `for` loop on lines 39 and 48. The number 3000 should instead be made into a static variable in order to make the iteration ceiling value more easily modifiable.

Maintenance Consequences?

1. Making the above change introduces another variable which adds more code to the solution.

7. Code fragment 7

```
public class BodyMass {
      public void calculateEverything(String name, double height,
             double weight, int age) {
            System.out.printf("%s's BMI is %.2f", name,
              (703 * weight / (height * height))
            );
      }
       public static void main(String[] args) {
            LongParams lp = new LongParams();
            lp.calculateEverything("John", 67, 150, 20);
      }
}
```

Smells and locations?

1. Long parameter list – line 2

Design principles?

1. The long parameter list code smell violates the design principle of information hiding. Instead of having four different parameters, this could be re-implemented to pass an object of type `Person` in order to reduce the amount of parameters passed.

Maintenance Consequences?

1. Creating another object as stated above could make it more tedious to obtain the data for each person. “Getter” methods will need to be implemented for the class, and it will be important for them to return the correct data type as well.

Name: Stamati Morellas COM S 362 - S20 Due 1/24/20 at 11:59pm
Section: 2

8. Code fragment 8

```
public class EventSchedule{
     public int month;
     public int date;
     public String day;
     public int hour;
     public int minute;
}
```

Smells and locations?

1. Primitive obsession – line 4

Design principles?

1. This smell violates the separation of concerns principle. The day of the week can be determined using the date (and month and year), so having a variable dedicated to this is unnecessary.

Maintenance Consequences?

1. A separate date class can be implemented to keep track of the date and time, however this takes a more dynamic approach as opposed to hard coding variables for the necessary values in the `EventSchedule` class, so it must be implemented with caution.

Name: Stamati Morellas COM S 362 - S20 Due 1/24/20 at 11:59pm
Section: 2

9. Code Fragment #9

```
public class PairedAverage{
     private int[] ar1 = {1,2,3,4,5};
     private int[] ar2 = {2,4,6,8,10};

     public void calculateEverything() {
          int ans1 = 1;
          int div1 = 0;

          for(int i=0;i<ar1.length;i++)
               ans1 *= ar1[i];
          div1 = ans1 / ar1.length;

          System.out.println("The average is " + div1);

          int ans2 = 1;
          int div2 = 0;

          for(int i=0;i<ar2.length;i++)
               ans2 *= ar2[i];
          div2 = ans2 / ar2.length;
          System.out.println("The average is " + div2);
     }
}
```

Smells and locations?

1. Duplicate code – lines 9-13, 18-21

Design principles?

1. This code smell violates the separation of concerns principle. Since the arrays (hard-coded) are the same size, there is no need to calculate the average on them more than once. This can all be done in the same place.

Maintenance Consequences?

1. If one array were to be larger than the other, there would be an error thrown if the averages were to be computed all at once. However, since the arrays are hard-coded, this should not be an issue.

Name: Stamati Morellas COM S 362 - S20 Due 1/24/20 at 11:59pm
Section: 2

10. Code Fragment #10

```
do {
        // input data source
        prompt("Data source? (0 = file, 1 = generated)");
        if (getInput() == 0L){

                //get data from file
                int[] fileMaster;

                //process data from file
                //input file name
                prompt("File name? ");
                String fname = getFileName();
                fileMaster = readFile(fname);

                expData = runSorts(fileMaster, sorters);

        } else {
                int[] randomData;

                // get experiment parameters
                prompt("How many values?");
                length = getInput();
                prompt("What seed? 0 = default ");
                seed = getInput();

                // generate random data
                randomData = new int[length];
                setRandomSeed(seed);
                for (int i = 0; i < length; i++){
                        randomData[i] = nextRandomInt();
                }
                expData = runSorts(randomData, sorters);
        }

        System.out.println(expData.formatTable());

        prompt("Another experiment? (0 = no, 1 = yes) ");
} while (getInput() == 1L);
```

Smells and locations?

1. Magic numbers – lines 4 and 38

Design principles?

1. This code smell violates the information hiding principle. Creating a constant variable to represent the right side of the operation in both cases would aid in the readability of the code and would ensure that the program behaves as it should. This program could unexpectedly terminate in certain scenarios where the user performs multiple experiments because the `while` loop does not depend on a constant value

Maintenance Consequences?

1. Incorrect refactoring could cause unexpected termination of the program depending on the set of inputs passed by the user.