Jeff Howes
Design Patterns - HW5 – Refactoring the Dungeon program

Details for the code refactoring that was implemented for Tom's Dungeon program.

1. Removed outdated keyboard class and method calls to bring the code up to date with the current version of the API. Refactored to use the newer Scanner method calls. The Dungeon, Hero, HeroFactory, Sorceress, Thief, and Warrior classes that required the new Scanner had the following import statement added: "import java.util.Scanner;"

   Offending code snippit:

```
public static boolean playAgain()
{
    char again;

    System.out.println("Play again (y/n)?");
    again = Keyboard.readChar();

    return (again == 'Y' || again == 'y');
}//end playAgain method
```

   Refactored code:

```
private static boolean playAgain()
{
    char again;
    Scanner input = new Scanner(System.in);

    System.out.println("Play again (y/n)?");
    again = input.nextLine().charAt(0);

    return (again == 'Y' || again == 'y');
}//end playAgain method
```

2. Created a Hero Factory to move the responsibility of creating heroes out of the Dungeons class, so the class will have higher cohesion (to just provide the high-level game play control). Offending code in the Dungeon class:

```
public static Hero chooseHero()
{
    int choice;
    Hero theHero;

    System.out.println("Choose a hero:\n" +
                       "1. Warrior\n" +
                       "2. Sorceress\n" +
                       "3. Thief");
    choice = Keyboard.readInt();

    switch(choice)
    {
        case 1: return new Warrior();

        case 2: return new Sorceress();

        case 3: return new Thief();

        default: System.out.println("invalid choice, returning Thief");
                 return new Thief();
    }//end switch
}//end chooseHero method
```

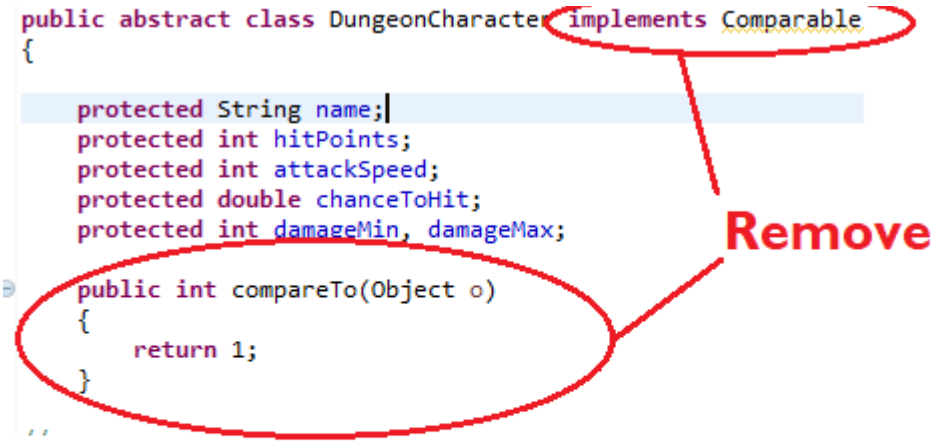
Refactored code:

```
package characters;
import java.util.Scanner;

public class HeroFactory {

    public static Hero createHero() {

        int choice;
        Scanner input = new Scanner(System.in);

        System.out.println("Choose a hero:\n" +
                           "1. Warrior\n" +
                           "2. Sorceress\n" +
                           "3. Thief");
        if(input.hasNextInt())
            choice = input.nextInt();
        else
            choice = 0;

        switch(choice)
        {
            case 1: return new Warrior();
            case 2: return new Sorceress();
            case 3: return new Thief();

            default: System.out.println("invalid choice, returning Thief");
                return new Thief();
        }
    }
}
```

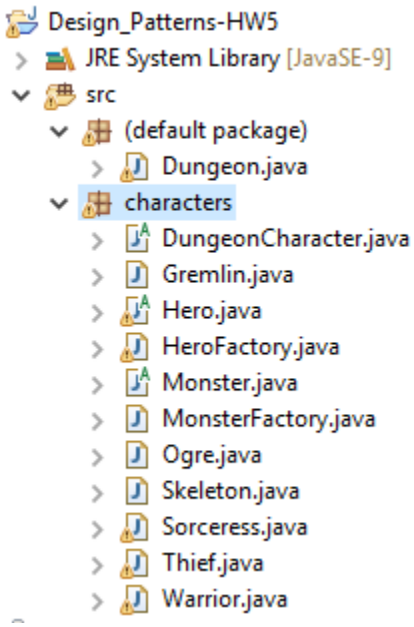
3. Created a Monster Factory (which randomly generate monsters) to move the responsibility of creating heroes out of the Dungeons class, so the class will have higher cohesion (to just provide the high-level game play control).
   Offending code is similar to createHero snippet above. Refactored code shown below.

```
package characters;

public class MonsterFactory {

    public static Monster createMonster() {

        int choice;
        choice = (int)(Math.random() * 3) + 1;

        switch(choice)
        {
            case 1: return new Ogre();
            case 2: return new Gremlin();
            case 3: return new Skeleton();

            default: System.out.println("invalid choice, returning Skeleton");
                     return new Skeleton();
        }
    }
}
```

4. Removed the “implements Comparable” and the compareTo(Object o) method from DungeonCharacter class, as it is dead code. The character compare feature was not utilized in the original program.



5. Organized all the Dungeon character classes and Factory classes into a character package. This was done to promote better organization, reuse and shareability of the character package, and to allow for package level protection/visibility on methods and fields that benefited from that level of protection.



6. Separated out the multiple responsibilities in each of the Heros' battleChoices() method. Moved the user menu & selection code into a usersBattleSelection() method and moved the attacking action code into the performAttack() method. This was to provide highly cohesive methods and classes and to provide methods that follow the single responsibility principal. Offending code shown below:

```
public void battleChoices(DungeonCharacter opponent)
{
    super.battleChoices(opponent);
    int choice = 0;

    do
    {
        choice = usersBattleSelection();
        performAttack(opponent, choice);

        numTurns--;
        if (numTurns > 0)
            System.out.println("Number of turns remaining is: " + numTurns);

    } while(numTurns > 0 && this.getHitPoints() > 0 && opponent.getHitPoints() > 0);

}//end overridden method

private void performAttack(DungeonCharacter opponent, int choice) {
    switch (choice)
    {
        case 1: attack(opponent);
            break;
        case 2: increaseHitPoints();
            break;
        default:
            System.out.println("invalid choice!");
    }//end switch
}
```

Refactored code:

```
private int usersBattleSelection() {
    int choice = 0;

    Scanner input = new Scanner(System.in);
    boolean validChoice;

    System.out.println("1. Attack Opponent");
    System.out.println("2. Increase Hit Points");
    System.out.print("Choose an option: ");

    do {
        if(input.hasNextInt() ) {
            choice = input.nextInt();

        }
        if(choice == 1 || choice == 2) {

            validChoice = true;
            break;
        }
        else
            validChoice = false;

                input.next();
                System.out.println("invalid choice!\n");
                System.out.println("1. Attack Opponent");
                System.out.println("2. Increase Hit Points");
                System.out.print("Choose an option: ");

    }while(!validChoice);
    System.out.println("");
  return choice;
}
```

7. Fixed the logic error in the Warrior, Thief, and Sorceress classes for better usability. An invalid attack selection from the user would decrement the hero's turn without the hero getting to attack.

Offending code: (showing bug)

```
public void battleChoices(DungeonCharacter opponent)
{
    super.battleChoices(opponent);
    int choice;

    do
    {
        System.out.println("1. Attack Opponent");
        System.out.println("2. Surprise Attack");
        System.out.print("Choose an option: ");
        choice = Keyboard.readInt();

        switch (choice)
        {
            case 1: attack(opponent);
                break;
            case 2: surpriseAttack(opponent);
                break;
            default:
                System.out.println("invalid choice!");
        }//end switch

        numTurns--;
        if (numTurns > 0)
            System.out.println("Number of turns remaining is: " + numTurns);

    } while(numTurns > 0);
```

Invalid choice (not a 1 or 2) would state "invalid choice!", but would not re-prompt user for a new selection. numTurns is just decremented.

Refactored code: battleChoices now continues to re-prompt user for a selection unit a valid choice is entered.

```
private int usersBattleSelection() {
    int choice = 0;

    Scanner input = new Scanner(System.in);
    boolean validChoice;

        System.out.println("1. Attack Opponent");
        System.out.println("2. Surprise Attack");
        System.out.print("Choose an option: ");

        do {
            if(input.hasNextInt() ) {
                choice = input.nextInt();

            }
            if(choice == 1 || choice == 2) {

                validChoice = true;
                break;
            }
            else
                validChoice = false;

                input.next();
                System.out.println("invalid choice!\n");
                System.out.println("1. Attack Opponent");
                System.out.println("2. Surprise Attack");
                System.out.print("Choose an option: ");

        }while(!validChoice);
        System.out.println("");
     return choice;
}
```

8. Refactored the battle method by removing the “print winner” section of code out of it and created a printWinner() method, so both the battle() and printWinner() method could follow the single responsibility principal.

Offending code:

```
public static void battle(Hero theHero, Monster theMonster)
{
    char pause = 'p';
    System.out.println(theHero.getName() + " battles " +
                       theMonster.getName());
    System.out.println("-----------------------------------------------");

    //do battle
    while (theHero.isAlive() && theMonster.isAlive() && pause != 'q')
    {
        //hero goes first
        theHero.battleChoices(theMonster);

        //monster's turn (provided it's still alive!)
        if (theMonster.isAlive())
            theMonster.attack(theHero);

        //let the player bail out if desired
        System.out.print("\n-->q to quit, anything else to continue: ");
        pause = Keyboard.readChar();

    }//end battle loop

    if (!theMonster.isAlive())
        System.out.println(theHero.getName() + " was victorious!");
    else if (!theHero.isAlive())
        System.out.println(theHero.getName() + " was defeated :-(");
    else//both are alive so user quit the game
        System.out.println("Quitters never win ;-)");

}//end battle method
```

9. Refactored the Dungeon's main method to use the template method Tom showed in class (template game pattern from the Template method lecture slides), for reusability, readability, and better high-level game control.

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

      Hero theHero;
      Monster theMonster;

      do
      {
          //Initialize the game
          theHero = HeroFactory.createHero();
          theMonster = MonsterFactory.createMonster();

          startGame(theHero, theMonster);

          do
          {
              battle(theHero, theMonster);
          }
          while(!EndOfGame (theHero, theMonster) );

          printWinner(theHero, theMonster);

      } while (playAgain() );
  }//end main method
```

10. Reduced many methods' and fields' visibility form public to protected or private for better encapsulation.

```
  private static boolean playAgain()
  {
      char again;
      Scanner input = new Scanner(System.in);

      System.out.println("Play again (y/n)?");
      again = input.nextLine().charAt(0);

      return (again == 'Y' || again == 'y');
  }//end playAgain method


  private static void startGame(Hero theHero, Monster theMonster) {
      System.out.println("");
      System.out.println(theHero.getName() + " battles " + theMonster.getName());
      System.out.println("-----------------------------------------");
  }

*-----------------------------------------------------------------
attle is the actual combat portion of the game.  It requires a Hero
nd a Monster to be passed in.  Battle occurs in rounds.  The Hero
oes first, then the Monster.  At the conclusion of each round, the
ser has the option of quitting.
-----------------------------------------------------------------*/
  private static void battle(Hero theHero, Monster theMonster)
  {
          //hero goes first
          theHero.battleChoices(theMonster);
```