Things You Should Know for Test 2

1. How to refactor procedural code into functions and classes.

2. How to trace programs containing functions and objects.

3. How to identify: signatures, constructors, head, body, definiton, member variables, member functions, method calls, parameters, and variable scopes.

4. What the difference between public and private is.

5. What the difference is between static functions of a class and methods of an object.

6. How to design your own classes.

7. How to overload a function.

8. How to override a function.

9. What the difference between pass by value and pass by reference is.

10. How to write a toString method and when that method is invoked by the compiler.

11. How to write setters and getters.

12. What encapsulation is and why we would organize programs into classes and objects.

13. What the scope of a variable is and how long various kinds of variables last for.

14. What the advantages of an Object-oriented program is over a procedural one.

15. How to use an existing object-oriented API.

Test 2: Functions, Classes, and Objects

Name______________________________

```
public class Armor{

  public Armor(Hero h){
    this.owner = h;
    this.shieldModifier = 1;
    this.name = "none";
  }

  public Armor(Hero h, int mod){
    this.owner = h;
    this.shieldModifier = mod;
  }

  public int shield(Monster m){
    return m.strength / shieldModifier;
  }

  public void equip(Hero h){
    owner = h;
  }

  public void unequip(){
    owner = null;
  }

  public boolean hasOwner(){
    return owner == null ? false : true;
  }

  private String name;
  private Hero owner;
  private int shieldModifier;
}


public class Weapon {

  public Weapon(Hero o){
    this.name = "bare hands";
    this.strengthModifier = 0;
    this.owner = o;
  }

  public Weapon(String name, int mod){
    this.name = name;
    this.strengthModifier = mod;
  }

  public String getName(){
    return name;
  }

  public int getStrengthModifier(){
    return strengthModifier;
  }


  public int use(Monster m){
    return
m.takeDamage(owner.getStrength() +
strengthModifier);
  }


  public void equip(Hero h){
    this.owner = h;
  }

  public Weapon unequip(){
    this.owner = null;
  }

  public boolean hasOwner(){
    return owner == null ? false : true;
  }

  private String name;
  private int strengthModifier;
  private Hero owner;
}
```

```
public class Monster{

  public Monster(String name, int maxHealth, int
strength){
    this.name = name;
    this.maxHealth = maxHealth;
    this.maxStrength = strength;
    this.health = health;
    this.strength = strength;
  }

  public int getHealth(){
    return this.health;
  }

  public int getStrength(){
    return this.strength;
  }

  public int getMaxHealth(){
    return this.maxHealth;
  }

  public int getMaxStrength(){
    return this.maxStrength;
  }

  public int attack(Hero h){
    return h.takeDamage(this.getStrength());
  }

  public int takeDamage(int amount){
    health -= amount;

    int damageAmount = amount;

    if(health < 0){
  damageAmount += health;
  health = 0;
    }

    return damageAmount;
  }

  public boolean isDead(){
    return health <= 0 ? true : false;
  }

  public int weaken(int value){
    this.strength -= value;

    int amountWeakend = value;

    if(strength < 0){
      amountWeakened += strength;
      strength = 0;
    }

    return amountWeakened;
  }

  public int strengthen(int value){
    this.strength += value;

    return value;
  }

  public int heal(int amount){
    health += amount;
    int amountHealed = amount;

    if(health > maxHealth){
      amountHealed -= (health - maxHealth);
      health = maxHealth;
    }

    return amountHealed;
  }

  private String name;
  private int maxHealth;
  private int health;
  private int strength;
  private int maxStrength;
}
```

```
public class Hero{

  public Hero(String name, int maxHealth, int
strength){
    this.name = name;
    this.maxHealth = maxHealth;
    this.health = maxHealth;
    this.strength = strength;
    this.maxStrength = strength;
    this.armor = new Armor(this);
    this.weapon = new Weapon(this);
    this.spell = new Spell(this);
  }

  public getName(){
    return this.name;
  }

  public getHealth(){
    return health;
  }

  public getStrength(){
    return strength;
  }

  public int getMaxHealth(){
    return maxHealth;
  }

  public int getMaxStrength(){
    return maxStrength;
  }

  public equipWeapon(Weapon w){
    this.dropWeapon();
    this.weapon = w.equip();
  }

  public equipArmor(Armor a){
    this.dropArmor();
    this.armor = a.equip();
  }

  public Weapon dropWeapon(){
    Weapon w = this.weapon;

    w.unEquip();
    this.weapon = new Weapon();

    return w;
  }

  public Armor dropArmor(){
    Armor a = this.armor;
    a.unEquip();
    this.armor = new Armor();

    return a;
  }

  public grantSpell(Spell s){
    this.spell = s;
  }

  public int castSpell(Monster m){
    return this.spell.cast(m);
  }

  public int castSpell(Hero h){
    return this.spell.cast(h);
  }

  public int attack(Monster m){
    return this.weapon.use(m);
  }

  public int takeDamage(int damage){
    return assignDamaged(armor.shield(damage));
  }

  private int assignDamage(int amount){
    int amountDamaged = amount;
    health -= amount;

    if(health < 0){
      amountDamaged -= (amount + health);
      health = 0;
    }

    return amountDamaged;
  }

  public boolean isDead(){
    return health <= 0 ? true : false;
  }

  public int heal(int amount){

    int amountAdded = amount;
    health += amount;


    if(health > maxHealth){
      amountAdded = amount - (health - maxHealth);
      health = maxHealth;
    }

    return amountAdded;
  }

  private String name;
  private int maxHealth;
```

```
  private int health;
  private int strength;
  private Spell spell;
  private Weapon weapon;
  private Armor armor;
  private int maxStrength;
}
```

Recognize Elements

Use the program on page 2 to answer the questions in this section.

1. How many classes are in the program listed?

`4, Armor, Weapon, Monster, Hero`

2. How many constructors does the Armor class have?

`2`

3. How many methods are in the Monster class?

`Armor(Hero), Armor(Hero, int)`

4. What is the signature of the use() method in the Weapon class?

`int use(Monster)`

5. Name the private member variables of the Hero class.

`name, maxHealth, health, strength, spell, weapon, armor, maxStrength`

6. What is the return type of the dropWeapon() method in the Hero class?

`Weapon`

7. On what classes does the Hero class depend?

`Armor, Weapon, Monster, and Spell`

8. Name the parameters of the Monster constructor.

`name, maxHealth, strength`

9. What does the method attack() of the Hero class do?

```
The attack method invokes the use method of the weapon object the Hero owns. This reduces the health of
a monster.
```

Trace Programs

See the DollChain add method. There is a mistake in the code, my apologies

```
public class RussianDoll{

  public RussianDoll(String name){
    this.name = name;
  }


  public RussianDoll(String name, RussianDoll r){
    this.name = name;
    this.r = r;
  }


  public RussianDoll getDoll(){
    return r;
  }


  public String getName(){
    return name;
  }


  public void insertDoll(RussianDoll r){
    this.r = r;
  }


  public String toString(){
    return this.name;
  }


  private String name;
  private RussianDoll r;

}
```

```
public class DollChain {

  public DollChain(RussianDoll r){
    this.outermost = r;
  }


  public DollChain(String name){
    this.outermost = new RussianDoll(name);
  }


  public void add(RussianDoll r){
    r.insertDoll(outermost);
    ~~outermost.insertDoll(r);~~ /* MISTAKE should have been outermost = r; */
    outermost = r; //new replacement code
  }

  public RussianDoll removeOutermost(){
    RussianDoll previousOutermost = this.outermost;

    this.outermost = this.outermost.getDoll();

    return previousOutermost;
  }

  public toString(){
    RussianDoll t = outermost;
    String s = “”;

    s += t.toString();

    while(t.getDoll() != null){
      t = t.getDoll();
      s += t.toString();
      s += “ “;
    }

    return s;
  }



  private RussianDoll outermost;
}
```

1.
```
RussianDoll first = new RussianDoll(“Bob”);
RussianDoll second = new RussianDoll(“Sally”);
RussianDoll third = new RussianDoll(“Tamara”);

DollChain c = new DollChain(first);
c.add(second);
c.add(third);

System.out.println(c);
```

OUTPUT:

```
Tamara Sally Bob
```

2.
```
RussianDoll a = new RussianDoll(“a”);
RussianDoll b = new RussianDoll(“b”);

a.insertDoll(b);
b.insertDoll(a);

System.out.println(a.getDoll().getDoll());
System.out.println(a.getDoll());
System.out.println(a);
```

```
a
b
a
```

Use an Existing API

```
public class Person{

  public Person(String name){
    population++;
    this.name = name;
  }


  public boolean isZombie(){
    return isZombie;
  }


  public static int getPopulation(){
    return population;
  }


  public String getName(){
    return name;
  }


  public void becomeZombie(){
    this.isZombie = true;
    population--;
    zombieCount++;
  }


  public void die(){
    if(!isZombie){
      population--;
    }
  }


  public void bite(Person p){
    if(this.isZombie){
      p.becomeZombie();
    }
  }


  private boolean isZombie;
  private static int population;
  private static int zombieCount;
  private String name;
}
```

Using the Person class, create 6 Person objects. Print out the population. Make one of the Person object become a zombie. Have that person who became a zombie bite one of the other people. Print out the population. Have one person who is not a zombie bite another person who is not a zombie. Print out the population.

```
Person anthony = new Person("Anthony");
Person bob = new Person("Bob");
Person cathy = new Person ("Cathy");
Person donna = new Person("Donna");
Person eric = new Person("Eric");
Person fran = new Person("Fran");

fran.becomeZombie();
fran.bite(bob);
System.out.println(Person.getPopulation());
anthony.bite(cathy);
System.out.println(Person.getPopulation());
```

Questions

1. What is encapsulation?

Encapsulation is a design principle in object-oriented programming in which code is organized by grouping together data with the operations performed on that data and then allowing that data to be manipulated and access only using the operations that belong to that unit. The allowed operations are called the unit's API or application programming interface. In Java, encapsulation is achieved by creating objects and making all of the properties of those objects (member variables) private. The programmer then writes public methods through which the variables can be accessed and manipulated.

2. What is the difference between pass by value and pass by reference?

Pass by value is how primitives like int, char, and double are passed as parameters to functions in Java. When variables are passed by value, a copy of the data that variable has is given to the function but not the variable itself. Pass by reference is how objects are passed as parameters to functions in Java. When objects are passed, the function has direct access to the original object and can manipulate its state via its public methods. When an object is changed inside a function, those changes persist in main, but when a primitive variable is changed inside a function, those changes only last for the life of the function since the data being manipulated is a copy of the original.

3. What is a constructor?

A constructor is a special method of a class that is used to construct or build an object of that class. It is the factory that creates members of a class.

4. What is a function signature?

A function signature is what Java uses to determine the unique identity of a function. It consists of the function's return type, its name, and the types of each parameter. (For example, int add(int,int))

5. What is the difference between a static method and an ordinary method?

A static method belongs to a class and so cannot refer to this since it does not belong to an object. Ordinary methods belong to an object and so each instance of a class has its own copy of that method with its own state.

6. Why is it a best practice to make our member variables of a class private?

This helps achieve the goal of encapsulation. Being strict about how our data is manipulated makes it easier to control the structure of our program and reduce errors from incorrect or unintended manipulations of the data.

7. What is the difference between a public and private method?

A public method can be used by an object of the class outside of that class. A private method can only be used inside the class itself for internal purposes.

8. What is the signature of a function?

A function signature is what Java uses to determine the unique identity of a function. It consists of the function's return type, its name, and the types of each parameter. (For example, int add(int,int))

9. What is the difference between a function's definition and a function's invocation or application (when you call it with the parentheses such as Math.pow(3,2))?

The definition describes how the function is going to work. It has a head describing how to use it and a body of code that will execute any time the function is invoked. Invoking a function or calling a function happens when another routine uses the function. This is the moment that the definition of the function actually executes inside a program.

10. When would you want to overload a function?

Function overloading is when two functions have the same name but a different signature. This could be useful if you want to help a programmer accomplish a task under diverse sets of circumstances. For instance, it makes sense to name a function that adds two number together, add. However, sometimes programmers may want to add two different types of numbers or more than just two numbers. In this case, instead of having to have multiple names for the same operation (add, sum, aggregate, etc) you can just name all of the functions add, and their different signatures will help the compiler distinguish between them. You could write a function add(int,int) that adds two integers and add(double,double) that adds two doubles or add(int,int,int) that adds three integers.

11. What is the return type of a function?

The return type is the type of the data that is being sent out of the function when it is finished executing

12. When you see the keyword new, what function is being called at this point?

The constructor of that class

13. What is the difference between a class definition and an instantiated object of the class?

A class definition is a blue print for creating an object. An object is an actual instance of that class with a state, life, and scope. Objects are created, manipulated, and destroyed. Classes persist until needed, do retain some state occasionally, but are mainly there to create new objects or provide utility functions.

14. What is the scope of a variable?

The space and time inside the program for which the variable endures. A private primitive variable lives from the point it is declared until the closing curly brace in which it is contained is set.

15. How long does a parameter passed into a function exist for?

A parameter that is one of the primitive types will exist only for as long as that particular call of the function is running. An objected passed into a function will exist at for the life of that function but will also likely continue as long as a variable in the outer program that called the function retains a variable referencing that object.

16. What is the difference between an Object type and a primitive type?

An object contains primitive variables and other objects as well as functions for manipulating those variables. A primitive type is just composed of data. Objects are passed by reference, primitives are passed by value.

17. What is the difference between the way objects and primitive variables are handled when they are passed as parameters to a function?

Functions are granted an actual reference to an object but are only given a copy of a primitive variable.

18. How many files can be in a project?

As many as your operating system or IDE will allow (many)

19. How many packages can be in a project?

As many as your operating system or IDE will allow (many)

20. How many files can be in a package?

As many as your operating system or IDE will allow (many)

21. How many classes can be in a file?

As many as your operating system or IDE will allow (many)

22. Under what circumstance is the toString() method implicitly called on an object?

When System.out.print or System.out.println are called or whenever the object needs to be typecasted to a String in order for an operation to work: `String greeting = “Good evening, “ + hero;`

23. When do you use the return type void?

Whenever you are writing a function that doesn’t return any data.

Design an Object-Oriented Program

Write an Ecosystem program. Besides the Ecosystem simulation class which is where your main function occurs, you have 2 other classes: a Deer and a Lion class. The Deer and Lion classes each have a private member variable for a name (to uniquely identify each object), an x coordinate, and a y coordinate (for the location of the object in the Ecosystem). The constructor for each of these classes takes an x, y, and name parameter as input to initialize the private member variables. The Lion and Deer classes also each have a void move(direction) method. When an animal moves, it increments or decrements its x or y coordinate depending on which direction it moved (north, south, east, or west). Additionally, the Lion class has a void eat(Deer) method. The Ecosystem class is in charge of simulating a set of 3 deer objects and 2 lion objects. The Simulation steps through 20 turns. In each turn, each Deer and Lion object moves a random direction (north, south, east, or west). When all moves for each organism have been made (in a single turn), the Ecosystem checks the location of each Lion and Deer to see if any Lion is also in the same space as a Deer. If a Lion is in the same space as a Deer, the Ecosystem has that Lion call its eat(Deer) method on the deer that is also occupying that coordinate. The Lion can only eat one Deer per turn. The process happens for each of the 20 steps of the simulation.

(SEE NEXT PAGE FOR ANSWER)

```
public class Deer {

  public Deer(String name, int x, int y){
    this.name = name;
    this.x = x;
    this.y = y;
  }


  public int getX(){
    return x;
  }


  public int getY(){
    return y;
  }


  public String getName(){
    return name;
  }


  void move(String direction){
    switch(direction){
      case "north":
        this.y++;
               break;

      case "south":
               this.y--;
               break;

      case "east":
               this.x++;
               break;

      case "west":
               this.x--;
               break;
    }
  }


  private String name;
  private int x;
  private int y;

}
```

```
public class Lion {

  public Lion(String name, int x, int y){
    this.name = name;
    this.x = x;
    this.y = y;
  }


  public int getX(){
    return x;
  }


  public int getY(){
    return y;
  }


  public String getName(){
    return name;
  }


  public void move(String direction){
    switch(direction){
      case "north":
        this.y++;
               break;

      case "south":
               this.y--;
               break;

      case "east":
               this.x++;
               break;

      case "west":
               this.x--;
               break;
    }
  }


  public boolean eat(Deer d){
    boolean justAte = false;

    if(d.getX() == this.x && d.getY() == this.y){
      System.out.println(name + " has eaten " + d.getName());
      justAte = true;
      this.hasEaten = true;
    }

    return justAte;
  }


  public boolean hasEaten(){
    return hasEaten;
  }


  public void makeHungry(){
    this.hasEaten = false;
  }


  private boolean hasEaten;
  private String name;
  private int x;
  private int y;

}
```

```
public class Ecosystem{

  public static void main(String [] args){

  Deer b = new Deer("b", 0, 0);
  Deer c = new Deer("c", 0, 1);
  Deer d = new Deer("d", 1, 0);
  Lion l1 = new Lion("l1", 0, -1);
  Lion l2 = new Lion("l2", -1, 0);

  System.out.println("Starting the simulation...");
  for(int i = 0; i < 20; i++){
    System.out.println("Processing ecosystem turn " + i +"...");

    System.out.println("Moving deer and lions around...");
    b.move(generateDirection());
    c.move(generateDirection());
    d.move(generateDirection());
    l1.move(generateDirection());
    l2.move(generateDirection());

    System.out.println("The first lion is hunting...");
    hunt(l1, b);
    hunt(l1, c);
    hunt(l1, d);

    System.out.println("The second lion is hunting...");
    hunt(l2, b);
    hunt(l2, c);
    hunt(l2, d);

    System.out.println("");
    System.out.println("");
  }


  }

  public static void hunt(Lion l, Deer d){
    if(isInSameSpace(l, d)){
      l.eat(d);
    }
  }

  public static boolean isInSameSpace(Lion l, Deer d){
    return l.getX() == d.getX() && l.getY() == d.getY();
  }

  public static String generateDirection(){
    int d = (int)Math.ceil(Math.random() * 4);
    String dir = "";

    switch(d){
      case 1:
        dir = "north";
        break;

      case 2:
        dir = "south";
        break;

      case 3:
        dir = "east";
        break;

      case 4:
        dir = "west";
        break;
    }

    return dir;
  }
}
```