SOLUTION
2D Arrays

1. Problems involving symmetry are common in computations involving art and the natural world. We will write a `mirrorRight` method that accepts a matrix and generates a new symmetrical one which contains a mirror image of the original, as if a mirror was placed along its right edge.

Sample Interactions:

```java
 Toolkit tk = new Toolkit();
 char[][] m1 = new char[][]{{'a','b','c'}, {'d','e','f'}, {'g','h','i'}};
 tk.print(m1) // Assume that Toolkit has a print() method which prints character arrays
 a b c
d e f
g h i
> char[][] result = tk.mirrorRight(m1, 3); // m1 is a 3 x 3 matrix
> tk.print(result)
 c b a
 f e d
 i h g
> char[][] m2 = new char[][]{{'a','b'}, {'c','d'}};
> result = tk.mirrorRight(m2, 2); // m1 is a 2 x 2 matrix
> tk.print(result)
 b a
d c
```

Answer:

```java
 public class Toolkit{
   /**
    * Input: 2D array of characters and a dimension n.
    * Assumes the array is an "n by n" matrix where n is >= 1.
    * Returns a new array which is a mirror image of the
    * original, with the mirror along the right edge.
    */
   public char[][] mirrorRight(char[][] matrix, int n){
     char[][] mirror = new char[n][n];
     for (int row = 0; row < matrix.length; row++){
       for (int col = 0; col < matrix.length; col++){
         mirror[row][n - col - 1] = matrix[row][col];
       }
     }
     return mirror;
   }
}
```
2. In this problem, we supply code for a musical instrument inheritance tree. We want to concentrate simply on the supertype-subtype relationships, so there are neither variables nor methods in the classes and interfaces.

```java
public interface Playable{}
public abstract class Instrument implements Playable{}
public abstract class StringedInstrument extends Instrument{}
public interface Smashable{}
public class Guitar extends StringedInstrument implements Smashable{}
public class BassGuitar extends Guitar{}
```

The following statement is valid:

```java
Guitar myGuitar = new BassGuitar();
```

(a) What is the static type of myGuitar? Guitar
(b) What is the dynamic type of myGuitar? BassGuitar

Indicate whether each declaration statement below is valid or invalid. Consider each one in isolation. In other words, assume that the DrJava interactions pane is reset before each statement is typed in. If you think the statement is invalid, explain (very briefly) why. If you think it is valid, write down the static and dynamic type of the variable.

(c) Guitar g = new Guitar();
   Invalid because:
   Valid with static type: Guitar \quad \text{dynamic type: } Guitar

(d) StringedInstrument s = new Guitar();
   Invalid because:
   Valid with static type: StringedInstrument \quad \text{dynamic type: } Guitar

(e) StringedInstrument s = new StringedInstrument();
   Invalid because: An abstract class can not be instantiated.
   Valid with static type: \quad \text{dynamic type: }:

(f) Instrument i = new Instrument();
   Invalid because: An abstract class can not be instantiated.
   Valid with static type: \quad \text{dynamic type: }:

(g) BassGuitar b = new Guitar();
   Invalid because: A Guitar is not a BassGuitar (not a subtype of BassGuitar)
   Valid with static type: \quad \text{dynamic type: }:

(h) Smashable s = new Smashable();
   Invalid because: An interface can not be instantiated.
   Valid with static type: \quad \text{dynamic type: }:
(i) Playable p = new Guitar();

Invalid because:
Valid with static type: Playable dynamic type: Guitar

(j) Smashable s = new BassGuitar();

Invalid because:
Valid with static type: Smashable dynamic type: BassGuitar
Database of Persons

3. Here we develop code for a database of Persons. A Person is either a Student or a FacultyMember. We supply code (which should not be modified) for Person, Student, and FacultyMember (see the reference handout).

(a) Write the Database class so that the interactions shown on the reference sheet work. It should have one constructor and a method called "report". Its state should include an array of Persons. The constructor takes an array of names. If a name begins with "Dr." then a FacultyMember with that name should be created, otherwise a Student with that name should be created. Hint: Use the String.startsWith() method; here is a demonstration of how it works:

```java
> String name = "Dr. Dave";
> name.startsWith("Dr.")
true
> name = "Dave";
> name.startsWith("Dr.")
false
```

For the report() method, it's important that each person's name and status (student/faculty) be printed. Exact formatting is not important (points will not be deducted for formatting differences). However, to match the format shown in the interactions, we suggest using System.out.print() to print the name and System.out.println() to print the status.

**Answer:**

```java
class Database{
    private Person[] people;
    public Database(String[] names){
        people = new Person[names.length];
        for (int i = 0; i < names.length; i++){
            if (names[i].startsWith("Dr")){
                people[i] = new FacultyMember(names[i]);
            } else {
                people[i] = new Student(names[i]);
            }
        }
    }

    public void report(){
        for (int i = 0; i < people.length; i++){
            if (people[i] != null){
                System.out.print(people[i].getName() + " - ");
                if (people[i] instanceof FacultyMember)
                    System.out.println("faculty");
                else if (people[i] instanceof Student)
                    System.out.println("student");
                else
                    System.out.println("unknown");
            }
        }
    }
}
```
(b) Create a method `getFacultyOrStudents` that takes a character as a parameter (assumed to be either 'f' for faculty or 's' for students). If the argument is 'f', it should return an array of all the faculty members, otherwise an array of all the students. The array should be the exact size as the number of faculty (if 'f' was the argument, likewise for students), and it should contain copies of the objects that were created in the constructor. You may create additional helper methods.

Answer:

```java
public Person[] getFacultyOrStudents(char t){
    int count = countPersons(t);
    Person[] results = new Person[count];
    int resultIndex = 0;
    for (int i = 0; i < people.length; i++){
        if (t == 'f' && (people[i] instanceof FacultyMember))
            results[resultIndex++] = new FacultyMember(people[i].getName());
        if (t == 's' && (people[i] instanceof Student))
            results[resultIndex++] = new Student(people[i].getName());
    }
    return results;
}

private int countPersons(char t){
    int count = 0;
    for (int i = 0; i < people.length; i++){
        if (people[i] == null)
            continue;
        if (t == 'f' && (people[i] instanceof FacultyMember))
            count++;
        if (t == 's' && (people[i] instanceof Student))
            count++;
    }
    return count;
}
```
Software Design

4. You’ve landed a job with a gaming company and have been asked to write code for a project that has already been roughly designed (see the reference handout).

(a) Draw an inheritance tree which corresponds to the design sketched out in the reference handout. Don’t write any code; there is no need to write down any variable or method names. Simply draw the inheritance tree. Use rectangles for concrete classes, ovals for abstract classes, and triangles for interfaces. Label each with a name (e.g. LifeForm) and the word "abstract", "concrete", or "interface".

If an entity can be an interface, then make it an interface. If it can be a subclass of another, then make it a subclass. NOTE: even though the keywords 'public' and 'abstract' do not appear in any of the declarations listed on the reference sheet, assume that you as the designer can designate them as such.
(b) For each method named below, if we wanted the inheritance tree to have the fewest number of concrete methods with that name, write down the class(es) which should contain such concrete methods. By "concrete method" we mean a method that has a body.

i. becomeEaten()
   Answer: LifeForm

ii. becomeDestroyed()
   Answer: Human, Dwelling

iii. eat()

(c) What is an advantage of defining a method as being abstract in a hierarchy rather than providing a concrete implementation of it?  

Answer:  
In general, abstraction allows one to work with the "big picture" in mind, without worrying about the details. It provides a "contract" - all concrete subclasses must have a concrete version of the method available to it (in its own class or in an ancestor’s class). Given an abstract method in an abstract class A or interface I, one can confidently write code in and for A and I that calls the abstract method.

(d) What is an advantage of creating a concrete (vs. abstract) method at the highest possible level?  

Answer:  
Reusability - it is written in one place (in an abstract or concrete class) and may be used by many subclasses. Maintainability and Readability - it is easier to debug, modify, and understand code that exists in one rather than many places.

(e) Based on your diagram, can a Sned eat itself? Why or why not?  
Yes. A Sned can eat a LifeForm, and a Sned is a LifeForm.

(f) Based on your diagram, can a Sned destroy itself? Why or why not?  
No. A Sned can destroy a Destroyable, but a Sned is not a Destroyable.