

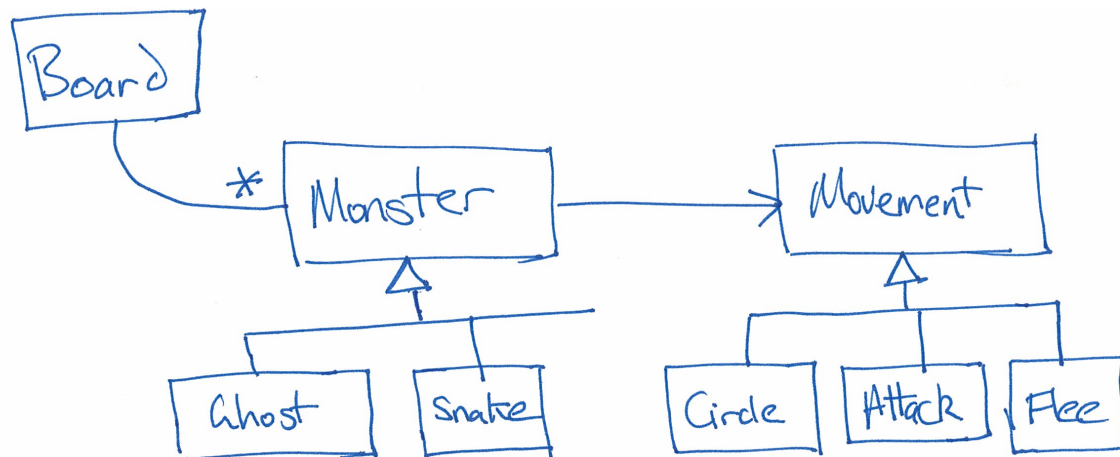
**EXAMINATIONS — 2010****END-OF-YEAR****SWEN222****Software Design****Time Allowed:** 2 Hours

Instructions: There are 120 possible marks on the exam.
Answer all questions in the boxes provided.
Every box requires an answer.
If additional space is required you may use a separate answer booklet.
Non-electronic Foreign language dictionaries are allowed.
Calculators ARE NOT ALLOWED.
No reference material is allowed.

Question	Topic	Marks
1.	Design Patterns	30
2.	Design By Contract	30
3.	Class Design	30
4.	API Design	30
Total		120

Question 1. Design Patterns

[30 marks]



The diagram shows part of the design of a 2-dimensional video game, like PAC-MAN. Every turn, each Monster (Ghost or Snake) has to move to a new square on the board. The Monster class decides where to move by asking a Movement object to plan its next move:

```

public abstract class Monster {

    // default movement is to move to attack the player
    private Movement movement = new Attack();

    public void move() {
        if (movement != null) {
            Location next = movement.planMove();
            this.animateMoveTo(next);
        }
    }
    // ...
}
  
```

(a) [5 marks] Which pattern is used to plan monsters' movements? Why would this pattern have been chosen here?

The STRATEGY Pattern. This pattern allows us to decouple the algorithm for determining movement from the Monster class. This makes it easier to provide different implementations of this algorithm, such as those listed (i.e. Circle, Attack, Flee).

Student ID:

Consider this specification of the `planMove()` method in the `Movement` interface:

```
interface Movement {  
    /**  
     * Plans the next move for a monster.  
     * @return the Location the monster will move to next.  
     */  
    Location planMove();  
}
```

(b) Consider an implementation of `planMove()` that returned a random board `Location`.

(i) [2 marks] Would this method meet the specification?

Yes, although I can see several possible problems:

- Presumably, `Monsters` cannot move into certain locations (e.g. walls)
- Presumably, `Monsters` must move to nearby locations and cannot jump to arbitrary locations.

(ii) [3 marks] Explain why you might want this behaviour in your program.

If you wanted to support `Monsters` which could teleport to arbitrary locations, it might make sense. It may also be helpful for testing the game.

(c) Consider an implementation of `planMove()` that simply returned the monster's current position.

(i) [2 marks] Would this method meet the specification?

Yes, although I can see the following problem:

- According to the description on page 2, monsters must move to a new location at each turn. The suggested implementation does not achieve this.

(ii) [3 marks] Explain why you might want this behaviour in your program.

This is perhaps the simplest possible implementation of the `planMove()` method. As such, it might be useful early on in the games' development as a placeholder, until more complex algorithms are implemented.

Student ID:

(d) [8 marks] Outline an implementation of the `Attack` class (mentioned on Page 2) which always moves the `Monster` towards the player's current `Location`.

```
class Attack implements Movement {
    private Board board;
    public Attack() {
        this.board = Board.getInstance(); // ugly use of static method
    }
    public Location planMove() {
        // first determine player's and my location
        Location player = board.getPlayerLocation();
        Location me;
        for (Monster m : board.monsters()) {
            Monster m = (Monster) c;
            if (m.getMovement() == this) {
                me = m.getLocation();
            }
        }
        // finally, determine next step
        return determineNextStep(player, me);
    }
    private Location determineNextStep(Location player, Location me) {
        ...
    }
}
```

(e) [7 marks] Discuss the design of the `Monster` class and `Movement` interface. Highlight their strengths and weaknesses.

Strengths:

- The `Movement` algorithm is decoupled from the `Monster` class. This makes it more flexible, and easier to extend with different kinds of movement.

Weaknesses:

- It's not clear how an instance of `Monster` can determine the current location of the player. In the above example, I have assumed a global static method — however, this is not ideal.
- It's not clear whether an instance of `Monster` actually stores its own current location. In the above example, I have assumed it does.
- The specification of the `planMove()` method contradicts the game description on page 2. In particular, the descriptions states that a monster must move to a new location, where as this is not specified in the `Movement` interface.

Student ID:

SPARE PAGE FOR EXTRA ANSWERS

Cross out rough working that you do not want marked.

Specify the question number for work that you do want marked.

Question 2. Design By Contract

[30 marks]

Consider the following implementation of a linked list, which compiles without error.

```
public class LinkedList<T> {
    private Link head = null;
    private int length = 0;

    public T get(int index) { return find(index).item; }

    public void set(int index, T item) { find(index).item = item; }

    public int length() { return length; }

    public void add(T item) {
        Link<T> ptr = head;
        if(ptr == null) {
            // empty list so append to head
            head = new Link<T>(item);
        } else {
            // non-empty list, so locate last link
            while(ptr.next != null) { ptr = ptr.next; }
            ptr.next = new Link<T>(item);
        }
        length++; // update length cache
    }

    // traverse list looking for link at index
    private Link<T> find(int index) {
        Link<T> ptr = head;
        int i = 0;
        while(i++ != index) { ptr = ptr.next; }
        return ptr;
    }

    private static class Link<S> {
        public S item;
        public Link<S> next;
        public Link(S item) { this.item = item; }
    }
}
```

(a) (i) [2 marks] What happens if the following piece of code is run?

```
new LinkedList<String>().get(1); A NullPointerException is thrown.
```

Student ID:

(ii) [5 marks] Write suitable *pre-* and *post-conditions* for each of the following methods.

```
int length() // ENSURES: \return >= 0 &&
              // forall 0<=i<=\return, get(i) is unchanged
```

```
T get(int index) // REQUIRES: length() > index
```

(iii) [4 marks] Briefly, discuss whether or not a `LinkedList` can be *cyclic* (i.e. contain a link which, either directly or indirectly, refers to itself).

No, the list cannot be cyclic. The reason is that when an item is added to the list, it is always added to the end. Only the `next` field of the last link is ever modified, and it is always assigned a new `Link`.

(iv) [5 marks] Give suitable *class invariants* for the `LinkedList` class. You may use written English if you prefer.

```
// INVARIANT: (head == null && length == 0) ||
               length = # Links reachable from head
// INVARIANT: no Link can reach itself (i.e. not cyclic)
```

Student ID:

(b) Ideas from the *functional programming paradigm* are commonly used to reduce the complexity of object-oriented software. Functional programming emphasises the use of methods which are *side-effect free*.

(i) [3 marks] Briefly, discuss what the term *side-effect free* means.

Side-effect free methods may not modify state which existed before they were run. They also cannot read or write I/O.

(ii) [5 marks] State which methods in `LinkedList` are side-effect free, and which are not. For those which are not, briefly state why.

- `get()` — is side-effect free.
- `set()` — is not side-effect free, since it may modify the state of a `Link`.
- `length()` — is side-effect free.
- `add()` — is not side-effect free, since it may modify the state of a `Link`.
- `find()` — is side-effect free

(iii) [6 marks] Briefly, discuss how you could convert `LinkedList` into the functional programming style (i.e. such that all methods are side-effect free).

I would modifier those methods which have side-effects (e.g. `add` and `set`) so that they did not modify **this** `LinkedList`. Instead, they would create a copy of **this** `LinkedList`, modify that and return it. e.g.

```
void set(int index, T item)
```

would become:

```
LinkedList<T> set(int index, T item)
```


Student ID:

SPARE PAGE FOR EXTRA ANSWERS

Cross out rough working that you do not want marked.

Specify the question number for work that you do want marked.

Question 3. Software Design

[30 marks]

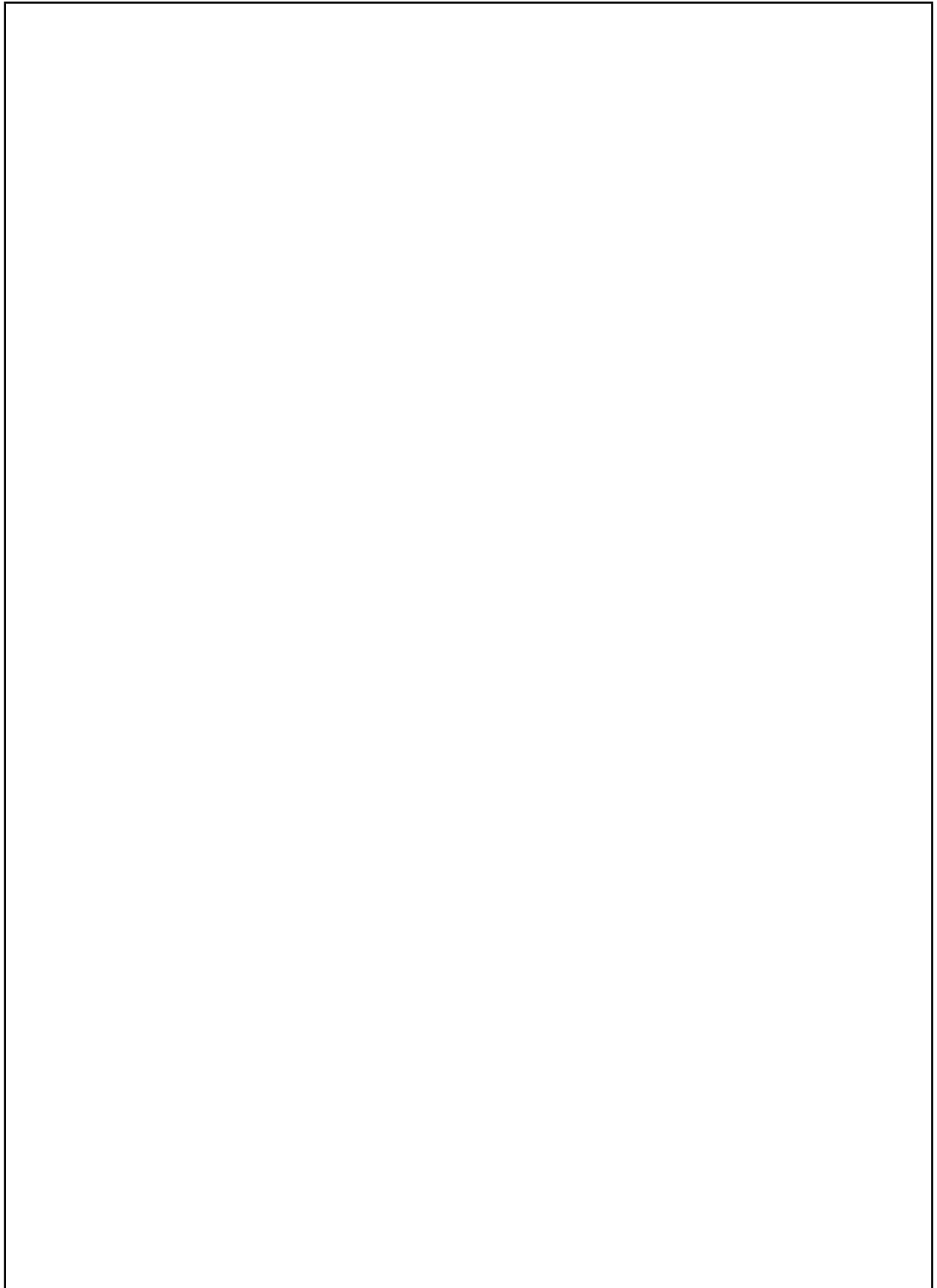
(a) This question deals with the design of an Adventure Game world, such as the game your group designed in the SWEN222 project.

An adventure game world is made up of *rooms*. Each room contains a set of *objects*, including *books*, *keys*, *candles* and more. Each object has a long and a short description. The player needs to be able to explore a room by listing the objects it contains and examining their descriptions. The player must also be able to pick up some objects, but not others (e.g. furniture cannot be picked up).

You may draw diagrams to explain your answers to any question, if you feel they are helpful.

(i) [15 marks] Briefly describe a simple design for implementing rooms which hold objects. To do this, list the main classes involved, including fields and methods as necessary to understand your design. For each class, briefly explain its purpose and justify any uses of inheritance, interfaces or polymorphism. **You do not need to give other implementation details.**

Student ID:

A large, empty rectangular box with a thin black border, occupying the central portion of the page. It is intended for the student to provide a response or show work.

Student ID:

(ii) [5 marks] Describe how your design allows a player to list the objects in a room and examine their descriptions.



(iii) [5 marks] Describe how your design allows a player to move from one room to the next. If your design does not allow this, describe how it could be extended to do so.



Student ID:

(iv) [5 marks] Describe how your design allows objects to be moved to different positions within a room. If your design does not allow this, describe how it could be extended to do so.

A large, empty rectangular box with a thin black border, intended for a student to draw or write their answer to the question above.

Question 4. API Design**[30 marks]**

Your company is designing the FunkiPlaya, a new music application like iTunes but better. This question is about the FunkiPlaya interface listed below:

```

/** A FunkiPlaya plays ``funky`` music. The data can be any kind of Music.
 * A FunkiPlaya has a playlist, and music can be added to the playlist,
 * or removed from it. shuffle() shuffles the current playlist.
 * snapshot() makes a new snapshot. The visualiser can be turned
 * on or off with the funkiVis() method. Finally, sync() can be used
 * to copy the playlist to an attached iPod or other portable device.
 */
public interface FunkiPlaya {
    /** Adds the Music to the current playlist.
     * @param music - music to add. */
    void getFunkie(Music music);

    /** Remove the Music matching name from the current playlist
     * @param Music - name that will be looked up in the playlist
     * @return Music stored with that name, or "MISSING" if not there. */
    Object delete(String name);

    /** Make a new screen snapshot and return it to the user.
     * (Calling snapshot twice writes contents of the playlist to disk)
     * @return a screen snapshot image */
    Object snapshot();

    /** Shuffle the current playlist.
     * @param funki - if TRUE, sort playlist in order of music name.
     *                - if FALSE, do a random shuffle.
     *                - if null, call snapshot() twice and delete the result.
     * @return TRUE if it is a weekend, FALSE otherwise */
    boolean shuffle(Object funki);

    /** Turn the visualiser on or off.
     * @param i - visualiser pattern to choose.
     * @return true if it worked */
    boolean get(int i);

    /** This syncs all information with an iPod.
     * @return Object array containing all the information. Even numbered
     * entries in this array (0,2,4) have a Music object which is in the
     * playlist. Odd numbered entries are: "DELETED" if that Music object
     * is missing (i.e. has been deleted); "DISK" if that Music is only
     * on disk; or an integer which is the number in the playlist
     * Warning: be very very careful if you update this array!
     * You might break the whole FunkiPlaya! */
    Object[] dump(); /* sync FunkiPlaya with iPod. May be slow */
}

```

Student ID:

(a) [12 marks] Circle and number six separate problems with the design of the FunkiPlaya interface on the previous page. For each problem, write a brief (i.e. one or two line) description of the problem in the corresponding box below.

1) `getFunki` — should be named `addToPlaylist` or similar.

2) Method `delete` is implements two operations (namely, looking up an item and deleting it)

3) Method `shuffle` has a ridiculous specification. The operation of this object should not depend on the which day of the week it is.

4) The documentation given for `get (int)` is poor. It is not clear how the integer `i` relates to a “visualiser pattern”.

5) Method `dump` clearly exposes its internal implementation, since updating the array should not be able to break the object.

6)
Method `dump ()` returns information in a very unstructured form. Instead of using an `Object []` array, a list of modifications would be preferable (e.g. `List<Modification>`). Here, each `Modification` would identify what operation was performed on which `Music` object.

Student ID:

SPARE PAGE FOR EXTRA ANSWERS

Cross out rough working that you do not want marked.

Specify the question number for work that you do want marked.

Student ID:

(b) [18 marks] In the box below, write a Java interface and Javadoc comments for your own replacement FunkiPlaya interface.

```
public interface FunkiPlaya {  
    /** Adds the Music to a given playlist  
     * @param music - music to add  
     * @param playlist - playlist to add music to  
     */  
    void add(Music music, Playlist playlist);  
  
    /** Looks up a Music object in a given playlist  
     * @param name - name to search for  
     * @param playlist - playlist to search  
     * @returns matching item, or null if none found  
     */  
    Music lookup(String name, Playlist playlist);  
  
    /** Deletes a Music object in a given playlist  
     * @param name - name to search for  
     * @param playlist - playlist to search  
     * @returns true if item deleted, false otherwise.  
     */  
    boolean delete(String name, Playlist playlist);  
  
    /** @returns the list of available playlists  
     */  
    List<Playlist> playlists();  
  
    /** Selects current playlist  
     * @param playlist - playlist to select as current  
     */  
    void select(Playlist playlist);  
  
    /** Shuffles the music in the current playlist  
     */  
    void shuffle();  
  
    /** Disables the Visualiser  
     * @returns true if successful, false otherwise.  
     */  
    boolean disableVisualiser();  
  
    /** Enables the Visualiser  
     * @param pattern - visualisation pattern to display  
     * @returns true if successful, false otherwise.  
     */  
    boolean enableVisualiser(VisualiserPattern pattern);  
}
```

Student ID:

SPARE PAGE FOR EXTRA ANSWERS

Cross out rough working that you do not want marked.

Specify the question number for work that you do want marked.

Student ID:

SPARE PAGE FOR EXTRA ANSWERS

Cross out rough working that you do not want marked.

Specify the question number for work that you do want marked.
