

G52OBJ-E1 Model Answers

Each questions answer begins on a new page...

Question 1

a) ii

b) v

c) i, iii, iv

d) iv

e) iii

f) iv

Question 2

a)

- Access to objects is provided by variables containing pointers to the objects in memory.
- Many variables (in different parts of the program) can point to the same object in memory
- It is therefore difficult to know at any one point in the program whether there are any other valid pointers to the object
- If we delete the object when there are still valid pointers in other parts of the program then the program will crash when those other parts try to access the object.

b)

- Reference counting moves the destruction of an object from being the role of the client to the role of the object. (1)
- The object is informed when pointers to it are created (e.g. by copying a pointer) or go out of scope...
- ...by calls to methods typically called `retain()` and `release()` (also can be builtin into the language, mark given if they indicate how it happens in either fashion)
- The object maintains a counter (called the *reference count*), of the number of active pointers to the object (1)
- everytime a new pointer to the object is created the counters is incremented and decrements this counter every time one goes out of scope. (2)
- When the reference count reaches zero, the object causes itself to be destroyed (e.g. in C++ by calling the `delete` operator)
- This happens as soon as the object goes out of scope
- In this way, the object is able to track how many pointers exist to it and know when it is safe to delete the object. (2)

c)

- Circular references is a problem that occurs when you have a cycle of objects pointing to each other. E.g. object A points to object B and object B points back at object A.
- In this case the two objects, A and B, will always have a reference count of at least 1 from the pointers to each other
- When the last pointer to object A (for example) is released, then the reference count of that object will drop to one (because B still holds a pointer to it) and object will not be destroyed (this happens only when the reference count equals zero)
- Both objects will still exist in memory at this point with no external pointers to either object and waste memory. There is no way to release the objects since the only pointers to the objects are

contained within the objects and they won't get released until the objects are destroyed (and one object won't be released until the other object is destroyed)

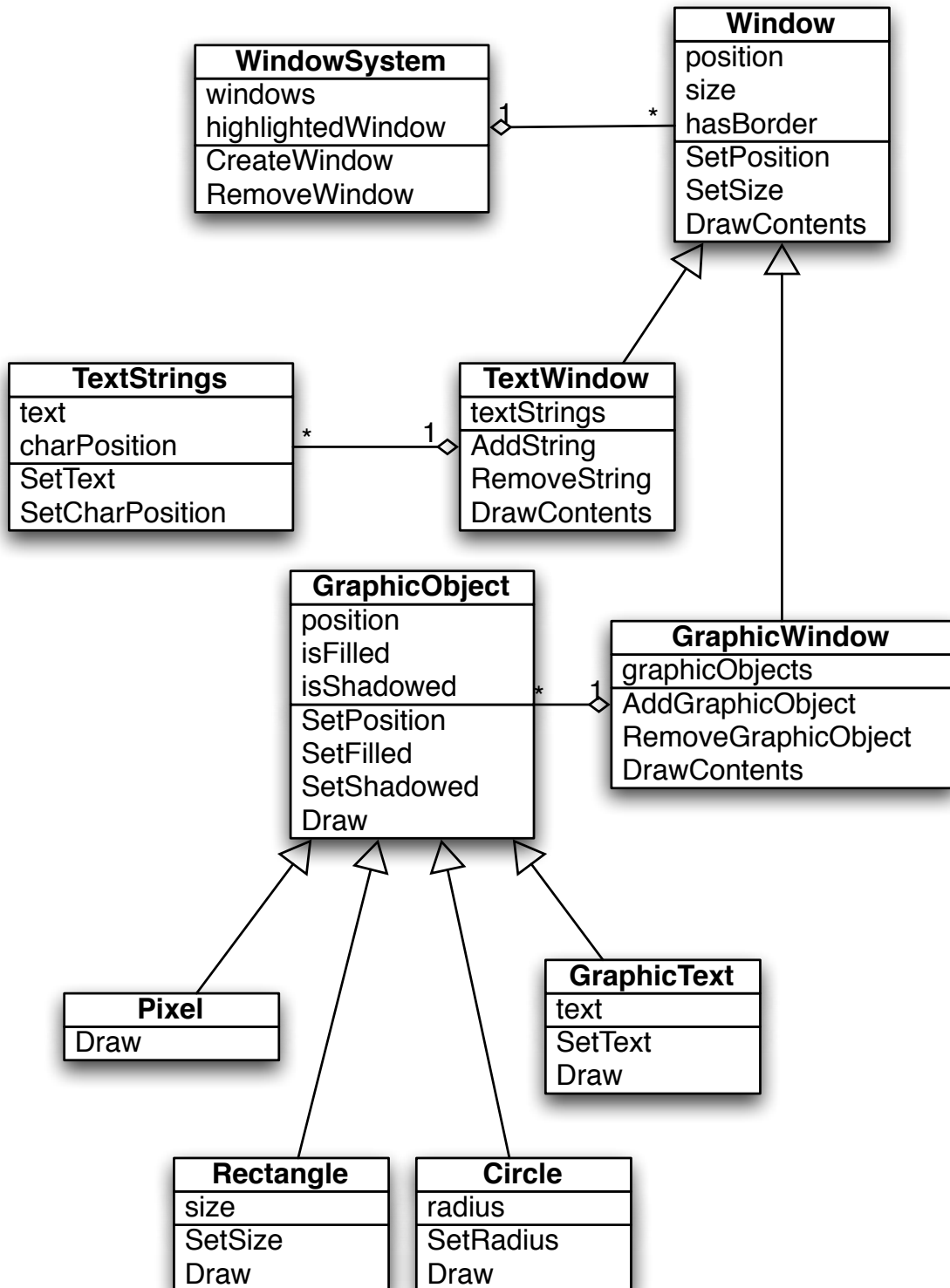
d)

- When reference counting is grafted on to an existing language such as C++, programmers must manually call **Retain()** and **Release()** each time they copy or set a pointer and when the pointers go out of scope.
- It is possible that the programmer may forget a call (or call in the wrong place) to **retain()** or **release()**
- This would mean the objects would either get destroyed too earlier (and cause a crash) or never get destroyed at all (wasting resources)

Q3)

- **Window System** — This class represents the whole window system maintaining a list of **Window** objects (one for each window) and a pointer to the currently highlighted **Window**.
- **Window** — base-class for the different types of window, provides the common functionality between the different window types (such as size, position on the screen and border).
- **TextWindow** — concrete window type that handles windows made up of text, maintains a collection of **TextString** objects that it draws within itself
- **TextStrings** — used by the **TextWindow** to contain strings. Each string has a position within the window.
- **GraphicWindow** — concrete window type that handles windows made up of graphics, maintains a collection of **GraphicObject** objects that it draws within itself
- **GraphicObject** — base-class for the various types of graphical objects, stores details off common features (position within the window, whether filled, or shadowed)
- **Pixel, Rectangle, Circle** — represents the corresponding Graphical type, sub-class of **GraphicObject**. Stores properties particular to that graphical object (e.g. radius)
- **GraphicText** — stores text for display in a graphical window. Cannot use **TextString** here as it is designed to work in a different coordinate system

b)



Question 4

a)

- The strategy pattern is used where we want to vary the algorithm that a class uses to perform some task.
- The algorithms are encapsulated into their own class.
- Main class uses an instance of one of these classes to perform the operation.
- All the algorithm classes must conform to a common interface otherwise the main class won't know how to call the object containing the algorithm
- Algorithm can be changed dynamically at run time.

b)

Expected result.

A good answer should exploit the fact that the only thing that changes is the comparison (marked helpfully by a comment) This can be moved off into a class of its own that compares staff Records. This will need an interface:

```
interface StaffRecordComparator
{
    public abstract boolean Compare(StaffRecord a, StaffRecord
b);
}
```

We need to add an instance variable to hold a reference to the comparator

```
private StaffRecordComparator m_comparator;
```

Initialize it in the constructor:

```
m_comparator = new StaffRecordCompareBySurname();
```

And provide a Mutator to change it

```
public void SetComparator(StaffRecordComparator c)
{
    if(c != null)
    {
        m_comparator = c;
    }
}
```

And finally change the code in `StaffList::Sort` (after the comment to):

```
if(m_comparator.Compare(m_staff[j],m_staff[j+1]))
{
    StaffRecord swapVar = m_staff[j];
    m_staff[j] = m_staff[j+ 1];
    m_staff[j+ 1] = swapVar;
}
```

All that is left is to implement the separate comparator objects:

```
public class StaffRecordSortBySurname implements
StaffRecordComparator
{
    public boolean Compare(StaffRecord a, StaffRecord b)
        {
            return (a.GetSurname().CompareTo(b.GetSurname()) > 0);
        }
}
```

etc;

Student need to demonstrate how **SetComparator** is called with a **SortByAge** comparator to get the list sorted by age