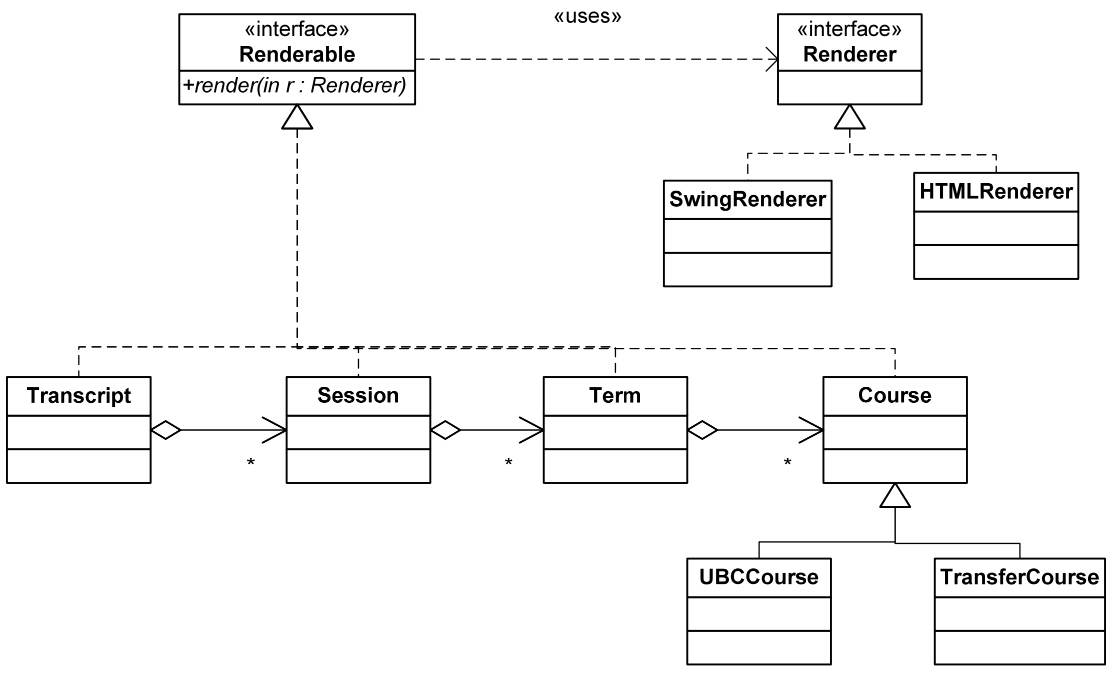
CPSC 211

SOLUTIONS to MIDTERM PRACTICE EXERCISES

<u>Software Design</u>

1. Although their may be other ways to design this problem, the following design is probably the simplest one.



2.
a)

	precondition	postcondition
`addItem`	*stronger*	*same*
`computeBill`	*same*	*stronger*
`checkOut`	*stronger*	*stronger*

b) `DeliveredGroceryOrder` is not a proper subtype of `GroceryOrder` according to the LSP because the preconditions on `addItem` and `checkout` are stronger. For the LSP to hold, preconditions in the subclass must be weaker (or the same) and postconditions must be stronger (or the same).

Exceptions

1. Output is:

```
catchit caught an exception
catchit caught an exception
In throwit a is 10
```

Software Testing

1.

a.
```
amount > 0
amount <= 0
```

b. Some sample test cases (others were possible):

```
1) theRow=25, theSeat = 50;    typical
2) theRow=1, theSeat = 1;      boundary
3) theRow=50, theSeat = 10;    boundary
4) theRow=49, theSeat = 1;     boundary
```

Java Collections etc.

1.

```
public static <E> void deleteAll(List<E> list,  E obj ){
  Iterator itr = list.iterator();

  while ( itr.hasNext() ) {
    if ( obj.equals( itr.next() ) )
      itr.remove();
  }
}
```

This operation take O(n) time in both cases. What if we don't use iterator and write:

```
public static <E> void deleteAll(List<E> list,  E obj ){

  for ( E cur : list ) {
    if ( obj.equals( cur ) )
      list.remove(cur);
  }
}
```

Is this a correct code?

2.
```
public static <E> List<Integer> getIndices(List<E> col,  E obj) {
  List<Integer> result = new ArrayList<Integer>();

  ListIterator itr = list.listIterator();

  while ( itr.hasNext() ) {
    if ( obj.equals( itr.next() )
      result.add(itr.previousIndex());
  }
  return result;
}
```

This operation take O(n) time.   What if we don't use a list iterator and use a for loop:

```
public static <E> void deleteAll(List<E> list,  E obj ){

  for ( int i = 0; i < list.size(); i++) {
    if ( obj.equals( list.get(i) ) )
      result.add(i);
  }
  return result;
}
```

What is the time for this?

3.
```
public static <E> Collection<E> removeDuplicates(Collection<E>
col) {
  return new HashSet<E>(col);
}
```

4.

a.
```
    public boolean equals( Object o )  {

        if ( o == null )
            return false;
        if ( getClass() != o.getClass() )
            return false;

        Dog d = (Dog) o ;
        return breed.equals(d.breed) && name.equals(d.name)
               && gender.equals(d.gender);
    }
```

b. Because has been overridden in this class, hasCode() nas to be defined as well to make it consistent with the equals() method.
The following components of the Dog class have to be used in the calculation of the hash code:

`bread` with value `breed.hashCode();`
`name` with value `name.hashCode();`
`gender` with value `genter.hashCode();`

5. The most efficient collection for this problem is a HashSet of dogs. Adding a dog and checking if a dog is registered are both O(1).