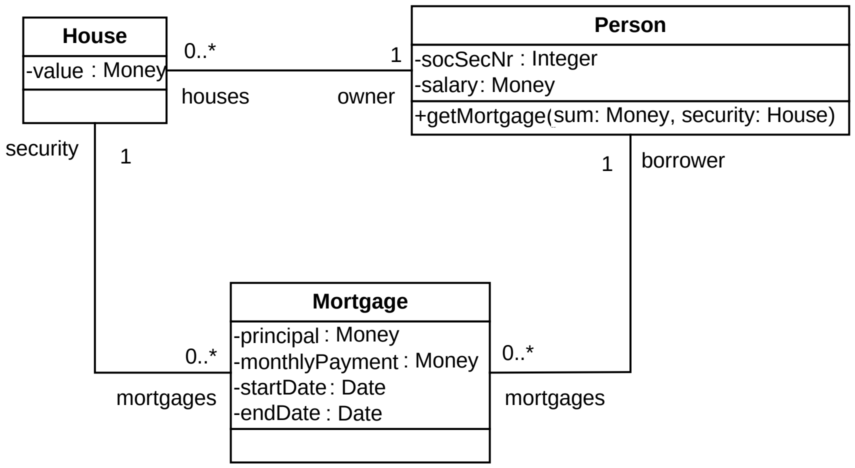
Carleton University
Department of Systems and Computer Engineering
SYSC-3120 Software Requirements Engineering Winter 2015

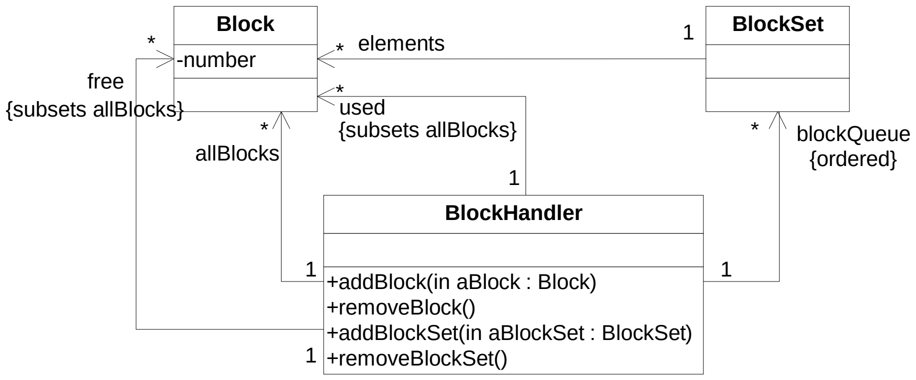
Assignment 2

Exercise 1 (10 marks)
Consider the class diagram below, write the following constraints with OCL:

- A person may have a mortgage on a house only if that house is owned by him or herself; one cannot obtain a mortgage on someone else's house.
- Conversely, may a house have mortgages, those mortgages are held by the owner of the house.
- The start date for any mortgage must be strictly before its end date. You will assume there exist a <<Datatype>> class called Date with an operation beforeThan(Date) which returns true if and only if the Date object on which it is called is strictly before (in the calendar) the Date object passed as a parameter.
- The social security number of all persons must be unique.
- Suppose class Person has an operation `getMortgageMonthlyPayment()` that returns (Integer value) the amount (in dollars only) of the monthly payments of all the mortgages of a person. What would be the postcondition of this operation describing the result value it returns?



Exercise 2 (10 marks)
An important part of a computer's operating system is the subsystem that maintains files created by users. Part of the filing subsystem is the block handler. Files in the file store are composed of blocks of storage that are held on a file storage device. During the operation of the computer, files will be created and deleted, requiring the acquisition and release of blocks of storage. To cope with this, the filing subsystem will maintain a reservoir of unused free blocks and keep track of blocks that are currently in use. When blocks are released from a deleted file they are normally added to the queue of blocks waiting to be added to the reservoir of unused blocks. (You will consider that "elements" is the rolename of the Block-BlockSet association, on the end of class Block.)



Note the use of the "subsets" constraint in the BlockHandler context. The collection of free blocks is a subset of the collection of all blocks, and the collection of used blocks is a subset of the collection of all blocks.

For the class diagram above, write the following constraints in OCL:

- No block will be marked as both unused and used.
- All the sets of blocks held in the queue (blockQueue) will be subsets of the collection of currently used blocks.
- The collection of used blocks and the collection of free blocks will be the total collection of blocks that make up files. In other words, a block for the all blocks collection is either free or used.
- The collection of free blocks will have no duplicate block numbers.

Exercise 3 (10 marks)
Reusing the context of the previous exercise, operation `removeBlockSet()` removes all blocks in the first element (a `BlockSet`) of `blockQueue` from the used blocks collection and adds them to the free blocks collection. Note that in order to remove a block set, the block queue should not be empty. Operation `addBlockSet()` adds a block set (parameter) to the block queue (at the end of the queue) and can add a block set only if all its elements are currently used.

Write the pre and post conditions for `removeBlockSet()` and `addBlockSet()` in OCL.

Exercise 4 (5 marks)
Again using the context of the two previous exercises, given the textual description and your answers to the previous exercise, can you comment on the multiplicities shown in the class diagram?

Exercise 5 (2 marks)
How would you stereotype the three classes of the class diagram above, i.e., which string would you put between << and >> when providing the class names in the class diagram? Justify.

Exercise 6 (5 marks)

Suppose we have the following hierarchies:

- class Food is a parent class for classes Meat and Grass;
- class Animal is a parent class for classes Carnivore and Herbivore

Additionally suppose that class Animal specifies two operations:

- void eat(Food)
- Animal offspring()

Suppose that Carnivore and Herbivore override those two methods:

- offspring() of a carnivore is a carnivore
- offspring() of an herbivore is an herbivore
- a carnivore eats Meat (not any Food)
- an herbivore eat Grass (not any Food)

Then consider the following method:

```
public Animal getChildAndFeedMother(Animal animal, Food food) {
  Animal child = animal.offspring();
  animal.eat(food);
  return child;
}
```

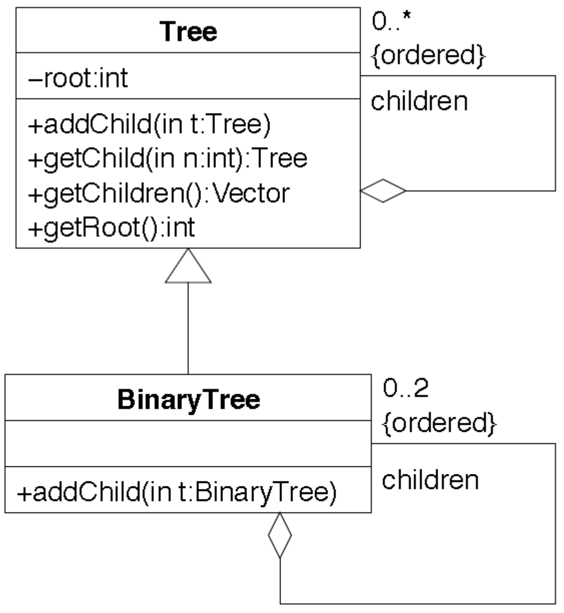
Explain whether the Liskov substitution principle holds or not.

Assignment 2

Exercise 7 (5 marks)

Below are the class diagram and source code for class Tree and the subclass BinaryTree. Do Tree and BinaryTree satisfy the Liskov Substitution Principle? Explain. If the Liskov principle is violated, list all the reasons.

```
import java.util.Vector;

public class Tree {
    /*
    * A tree is represented with <value, [children]>, where value is the int value of the root of the
    * tree and children is a sequence of zero or more Tree objects that are the children of this tree
    * node. A tree may not contain cycles, and may not contain the same tree object as a sub-tree
    * in more than one place.
    */

    // The root value
    private int root;
    // Vector of tree objects representing the children from the root
    private Vector children;

    // Creates a tree with root = val and no children: <val, [ ]>
    public Tree (int val) {
        root = val;
        children = new Vector();
    }

    // Adds t to the children of this, as the rightmost child
    public void addChild (Tree t) {
        children.add(t);
    }

    // Returns the Tree that is the nth leftmost child of this
    public Tree getChild (int n) {
        if ((n>0) && (n<=children.size())) {
            return (Tree)children.get(n-1);
        } else
            return null;
    }

    public Vector getChildren() {
        return children;
    }

    public int getRoot() {
        return root;
    }
}
```



```
public class BinaryTree extends Tree {
    /*
    * A BinaryTree is a tree where the nodes are int values and each node has zero, one or two
    * children. A typical BinaryTree is <value, [children]> where value is the int value of the root of
    * the tree and children is a sequence of zero, one or two BinaryTree objects that are the
    * children of this tree node. A BinaryTree may not contain cycles, and may not contain the
    * same BinaryTree object as a sub-tree in more than one place.
    */

    // Creates a tree with root = val and no children: <val, [ ]>
    public BinaryTree (int val) {
        super(val);
    }

    // Adds t to the children of this, as the rightmost child if t has at most one child
    public void addChild (BinaryTree t) {
        if (getChildren().size()<2) {
            t.addChild((Tree)t);
        }
    }
}
```