420-201-RE Assignment 3-A Due date: April 3, 2017
Java Collection Classes

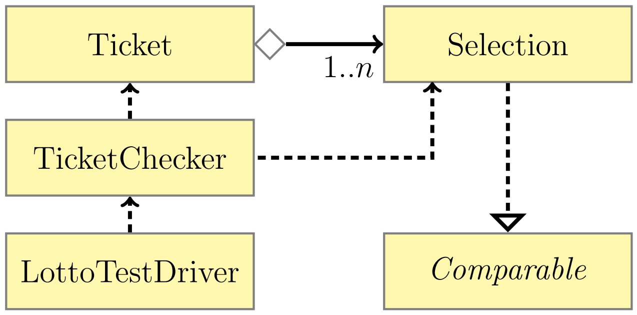
1 Objectives

- Practice using Java Collection classes
- Use interfaces, rather than inheritance, to make polymorphic calls.

2 Your Task

Write a program that simulates a lottery ticket and a lottery ticket checker machine.

Your program should be structured as shown by the class diagram at right. The specifics of the classes shown in the diagram are listed below:



3 Class Selection

Models a *set* of specified number of *unique* integers selected randomly from a specified range. For example:

```java
// a selection set of 9 unique random integers in the range 1 through 67
Selection lotto967 = new Selection(9, 1, 67);
System.out.println(lotto967);

// a selection set of 6 unique random integers in the range 1 through 49
Selection lotto649 = new Selection(6, 1, 49);
System.out.println(lotto649);
```

Sample output

```
12 16 24 35 43 54 55 56 64
10 15 24 26 33 45
```

Class **Selection** should include the following fields and methods:

3.1 Instance Fields in Class Selection

size Stores the size of this selection set.

low Stores the lowest possible random integer.

high Stores the highest possible random integer.

selection Stores a **Set<Integer>** reference to a **TreeSet<Integer>** object that will hold the random integers in this set.

Recall that **Set**s do not allow duplicates and that **TreeSet**s store their contents *sorted*: exactly what we want for the numbers in a selection set: unique and sorted. Lucky for us, a **TreeSet** does all that and more for us for free! All we need to do is add numbers to the set, and **TreeSet** will take care of the nitty-gritty of keeping the numbers in the set unique and sorted.

3.2 Instance Methods in Class Selection

public Selection(int size, int low, int high)
Constructs **this** new object, initializing it according to the supplied parameters.

public int compareTo(Object obj) Implements the only method specified by the **Comparable** interface (see class diagram shown on page 1 and here). Compares the contents of **this** and **other** selection sets using the folowing algorithm:

```
1  Type cast obj to a Selection named other.
2  if size of this is less than size of other then
3      Return $-1$
4  else if size of this is greater than size of other then
5      Return $+1$
6  Initialize an iterator other_it to scan the other selection
7  Initialize an iterator this_it to scan the this selection
8  while there are unvisited elements in this selection do
9      if the integer referenced by this_it is less than the integer referenced by
         other_it then
10         Return $-1$
11     else if if the integer referenced by this_it is greater than the integer
         referenced by other_it then
12         Return $+1$
13 Return 0
```

private void generate()
Generates **size** unique random integers in the range **low** through **high**, storing them in the **selection** set.

public int getLow() Returns **low**.

public int getHigh() Returns **high**.

public int getSize() Returns **size**.

public int match(Selection winner)
Returns the size of the intersection of **this** and **winner**'s selection sets.

public String toString() Return a string representation of this selection as shown in the examples above.

3.3 Class Fields in Class Selection

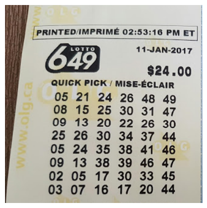
None.

3.4 Class Methods in Class Selection

None.

4 Class Ticket

This class models a ticket that has a specified number of selection (or play) sets. Each set costs $3. For example, the sample image at right shows a **649** ticket with 8 selection sets.



```
PRINTED/IMPRIMÉ 02:53:16 PM ET
LOTTO 649                 11-JAN-2017
                              $24.00
QUICK PICK / MISE-ÉCLAIR
05 21 24 26 48 49
08 15 25 30 31 47
09 13 20 22 26 30
25 26 30 34 37 44
05 24 35 38 41 46
09 13 38 39 46 47
02 05 17 30 33 45
03 07 16 17 20 44
www.olg.ca
```

Class Ticket should have the following fields and methods:

4.1 Instance Fields in Class Ticket

selection_size Stores the size of a selection set.

low Stores the lowest possible random integer.

high Stores the highest possible random integer.

ticket_size Stores the number of selection sets on this ticket.

selection_list Stores a **List<Selection>** reference to a **LinkedList<Selection>** object that will hold **selection_sets** selections in this ticket. Recall that **LinkedList**s do allow duplicates and that **LinkedList**s store their contents *ordered* (as opposed tp *sorted*).

date Stores the date this ticket was created.

4.2 Instance Methods in Class Ticket

public Ticket(int ticket_size, int selection_size, int low, int high)
Constructs **this** new **Ticket** object, initializing it according to the supplied parameters. For example:

```java
Ticket lucky = new Ticket(5, 6, 1, 49);
System.out.println("Ticket with ordered selections:");
System.out.println(lucky);
```

output

```
Ticket with ordered selections:
Sun Mar 19 14:00:54 EDT 2017
selection size  : 5
selection Range : [1,49]
ticket size     : 5
ticket cost     : $15.0
=================
04 19 23 25 27 31
09 11 19 27 35 45
14 27 28 30 36 48
09 13 14 21 33 44
01 07 13 26 41 46
=================
```

public Iterator<Selection> getListIterator()
Returns an **iterator** to scan the **selection_list** on this ticket.

public int getLow() Returns **low**.

public int getHigh() Returns **high**.

public int getSelectionSize() Returns **size**.

int getTicketSize() Returns number of selection on this ticket.

public double getSelectionCost() Returns the cost of one selection.

public double getTicketCost() Returns the cost of this ticket.

public String toString() Return a string representation of this selection as shown in the example above.

public String toStringSorted() Same as **toString()** above, except that the selections on this ticket are sorted. For example:

```java
System.out.println();

System.out.println("Tickets with sorted selections:");
System.out.println(lucky.toStringSorted());
System.out.println();
```

output

```


Tickets with sorted selections:
Sun Mar 19 14:00:54 EDT 2017
selection size  : 5
selection Range : [1,49]
ticket size     : 5
ticket cost     : $15.0
=================
01 07 13 26 41 46
04 19 23 25 27 31
09 11 19 27 35 45
09 13 14 21 33 44
14 27 28 30 36 48
=================
```

4.3 Class Fields in Class Ticket

None.

4.4 Class Methods in Class Ticket

None.

5 Class TicketChecker

This class models a ticket checker device that takes a ticket as input, scans the ticket extracting all the selection sets on it, and finally displays a message telling the ticket owner their prize amount, zero or more dollars!

In the class diagram on page 1, the dotted arrow lines from class **TicketChecker** to classes **Ticket** and **Selection** indicate that a **TicketChecker** object does not internally store instance references to **Ticket** and **Selection**; it rather *uses* or *depends on* them locally in a method as variables, or as arguments in method calls, etc.

Class **TicketChecker** should include the following components:

5.1 Class Fields in Class TicketChecker

winning_selection A private static **Selection** reference initialized to **null**.

5.2 Class Methods in Class TicketChecker

public static double PrintPrize(Ticket ticket)

If **winning_selection** is **null**, the method sets it to a reference to a newly created **Selection** object. That is, **winning_selection** is set once during a program run. In the real world, the device reads current winning numbers from a server.

The method intersects the winning selection set with each of the selection sets on **ticket**, displaying information about the number of matches m on each set, and showing the prize amount for each set.

For the sake of simplicity, it computes the prize as follows: if $m > 0$, then $prize = \$10^m$; otherwise, $prize = \$0$.

For Example:

```java

TicketChecker.PrintPrize(lucky);
System.out.println("Thank you for playing and for keeping us rich!");
```

```
output


winning selection: 07 14 36 44 48
selection: 04 19 23 25 27 31     matches: 0  prize: $0.00
selection: 09 11 19 27 35 45     matches: 0  prize: $0.00
selection: 14 27 28 30 36 48     matches: 3  prize: $1000.00
selection: 09 13 14 21 33 44     matches: 2  prize: $100.00
selection: 01 07 13 26 41 46     matches: 1  prize: $10.00
                                       Total prize: $1110.00
Thank you for playing and for keeping us rich!
```

5.3 Instance Fields in Class TicketChecker

None.

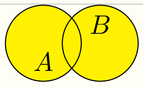
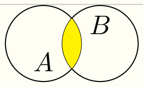
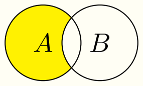
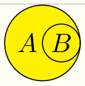
5.4 Instance Methods in Class TicketChecker

None.

6 Class LottoTestDriver

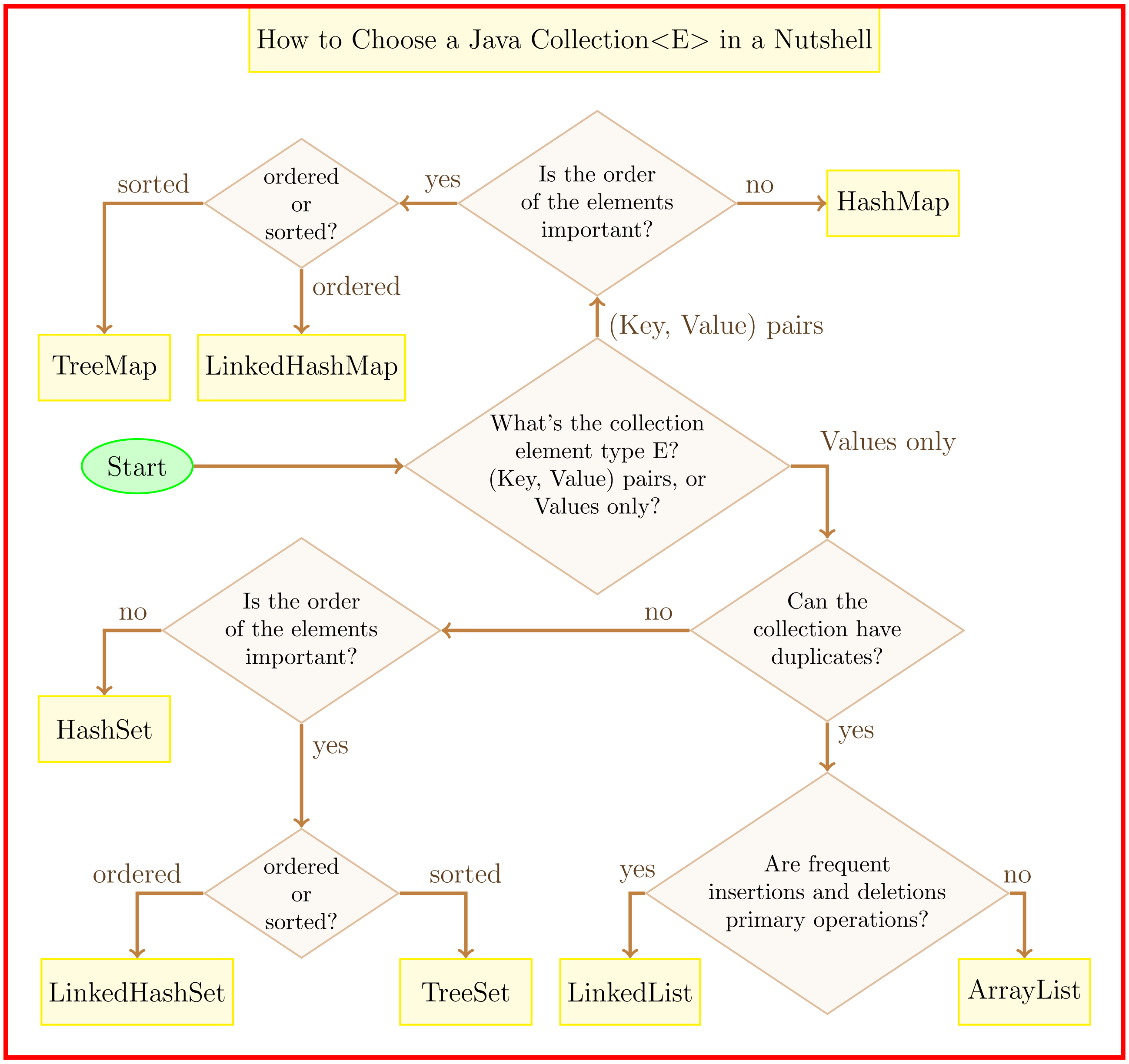
```java
public class LottoTestDriver
{
  public static void main(String[] args)
  {
     Ticket lucky = new Ticket(5, 6, 1, 49); // 5 plays of lotto 6/49

// print the selection sets in the order that they were generated
     System.out.println("Ticket with ordered selections:");
     System.out.println(lucky);
     System.out.println();

// print the selection sets in ascending order of selection sets
     System.out.println("Tickets with sorted selections:");
     System.out.println(lucky.toStringSorted());
     System.out.println();

     TicketChecker.PrintPrize(lucky); // do we feel lucky!?
     System.out.println("Thank you for playing and for keeping us rich!");
  }
}
```

Common Set Operations, Given Two Sets A and B

Set operation	Java	Description	Venn Diagram
Union	`A.addAll(B)`	Set of all elements that are in A, B, or both	
Intersection	`A.retainAll(B)`	Set of all elements that are in both A and B	
Difference	`A.removeAll(B)`	Set of all elements that are in A but not in B	
Superset, Subset	`A.containsAll`(B)	Returns true if A is a superset of (contains all elements of) B	

Java Collection Classes in a Nutshell

		Implementations				
		Hash Table	Resizable Array	Balanced Tree	Linked List	Hash Table & Linked List
Interfaces	Set	`HashSet`		`TreeSet`		`LinkedHashSet`
	`List`		`ArrayList`		`LinkedList`	
	`Queue` `Deque`		`ArrayDeque`		`LinkedList`	
	`Map`	`HashMap`		`TreeMap`		`LinkedHashMap`


How to Choose a Java Collection<E> in a Nutshell
Start
What's the collection element type E? (Key, Value) pairs, or Values only?
(Key, Value) pairs
Values only
Is the order of the elements important?
yes
no
HashMap
ordered or sorted?
sorted
ordered
TreeMap
LinkedHashMap
Can the collection have duplicates?
no
yes
Is the order of the elements important?
no
HashSet
yes
ordered or sorted?
ordered
sorted
LinkedHashSet
TreeSet
Are frequent insertions and deletions primary operations?
yes
no
LinkedList
ArrayList

Vanier College, 420-202-RE, Winter 2017

Evaluation Criteria	
Correctness of execution of your program	60%
Proper use of required Java concepts	20%
Java API documentation style	10%
Comments on nontrivial steps in code, Choice of meaningful variable names, Indentation and readability of program	10%