Understanding Loops

```
/**
 * Determine if the collection generated by the given Traversal
 * contains an element that satisfies the given predicate.
 */
public <T> boolean contains(Traversal<T> tr, Predicate<T> choice){
  try{
    if (tr.isEmpty())
      return false;
    else
      if (choice.select(tr.getFirst()))
        return true;
      else
        return contains(tr.getRest(), choice);
  }
  catch(IllegalUseOfTraversalException e){
    System.out.println("Illegal traversal: " + e.getMessage());
    return false;
  }
}

/**
 * Count how many elements in the collection generated by the
 * given Traversal satisfy the given predicate.
 */
public <T> int countSuch(Traversal<T> tr, Predicate<T> choice){
  try{
    if (tr.isEmpty())
      return 0;
    else
      if (choice.select(tr.getFirst()))
        return 1 + countSuch(tr.getRest(), choice);
      else
        return countSuch(tr.getRest(), choice);
  }
  catch(IllegalUseOfTraversalException e){
    System.out.println("Illegal traversal: " + e.getMessage());
    return 0;
  }
}
```

```
/*----------------------------------------------------------------------------
   TEMPLATE - ANALYSIS:
   --------------------
   ReturnType method-name(Traversal<T> tr){
                            +--------------------+
   // invoke the methodAcc: | acc <-- BASE-VALUE |
                            +--------------------+
      method-name-acc(Traversal<T> tr, BASE-VALUE);
    }

    ReturnType method-name-acc(Traversal<T> tr, ReturnType acc)
    ... tr.isEmpty() ...                           -- boolean     ::PREDICATE
    if true:
    ... acc                                     -- ReturnType     ::BASE-VALUE
    if false:
       +---------------+
    ...| tr.getFirst() | ...                        -- T          ::CURRENT
       +---------------+

    ... update(T, ReturnType)                   -- ReturnType     ::UPDATE
             +----------------------------+
    i.e.: ...| update(tr.getFirst(), acc) | ...
             +----------------------------+
        +--------------+
    ... | tr.getRest() |                         -- Traversal<T>  ::ADVANCE
        +--------------+

    ... method-name(tr.getRest(), ReturnType)    -- ReturnType
    i.e.: ... method-name-acc(tr.getRest(), update(tr.getFirst(), acc))
/-----------------------------------------------------------------------------

    COMPLETE METHOD TEMPLATE:
    -------------------------
    <T> ReturnType method-name(Traversal<T> tr){
                             +----base-value------+
    // invoke the methodAcc: | acc <-- BASE-VALUE |
                             +--------------------+
      method-name-acc(Traversal tr, BASE-VALUE);
    }

    <T> ReturnType method-name(Traversal<T> tr, ReturnType acc){
          +---predicate--+
      if (| tr.isEmpty() |)
          +--------------+
       return acc;
     else
                              +----advance---+  +----update-using-current----+
       return method-name-acc(| tr.getRest() |, | update(tr.getFirst(), acc) |);
                              +--------------+  +----------------------------+
     }

     <T> return-type update(T t, return-type acc){
     ...
     }
```

```
/-----------------------------------------------------------------------
 orMap:
 boolean orMap(Traversal tr, Predicate choice){
   return orMapAcc(tr, false, Predicate choice);
 }

 Method Header:  boolean orMapAcc(Traversal tr, boolean acc, Predicate choice)
 BASE-VALUE:     false
 UPDATE:         boolean update(T t, boolean acc, Predicate choice){
                   return (choice.select(t)) || acc;
                 }
 -----------------------------------------------------------------------*/

/** RECURSIVE VERSION
 * Determine if any data item generated by the given traversal
 * satisfies the given Predicate predicate.
 */
public <T> boolean orMap(Traversal<T> tr,
                         Predicate<T> choice){

  return orMapAcc(tr, false, choice);
}

/** RECURSIVE VERSION --- accumulator based helper.
 * Determine if any data item generated by the given traversal
 * satisfies the given Predicate predicate.
 */
public <T> boolean orMapAcc(Traversal<T> tr,
                            boolean acc,
                            Predicate<T> choice){
  if (tr.isEmpty())
    return acc;
  else
    return orMapAcc(tr.getRest(),
                    updateOrMap(tr.getFirst(), acc, choice),
                    choice);
}

/** A helper to produce the updated value of the accumulator
 * @param <T> the type of data in this data set
 * @param t the instance of the data to be used in the update
 * @param acc the current value of the accumulator
 * @param choice the given Predicate predicate.
 * @return the updated value of the accumulator.
 */
protected <T> boolean updateOrMap(T t,
                                  boolean acc,
                                  Predicate<T> choice){
  return (choice.select(t)) || acc;
}
```

```
/*---------------------------------------------------------
countSuch:
int countSuch2(Traversal<T> tr, Predicate<T> choice){
  return countSuchAcc(tr, 0, choice);
}

Method Header:  int countSuchAcc(Traversal tr, int acc, Predicate choice)
BASE-VALUE:     0
UPDATE:         int update(T t, int acc, Predicate choice){
                 if (choice.select(t))
                   return acc + 1;
                 else
                   return acc;
                }
------------------------------------------------------------------*/

/** RECURSIVE VERSION
 * Count how many data elements generated by the given traversal
 * satisfy the given Predicate predicate.
 */
public <T> int countSuch2(Traversal<T> tr, Predicate<T> choice){
  return countSuchAcc(tr, 0, choice);
}

/** RECURSIVE VERSION --- accumulator based helper.
 * Count how many data elements generated by the given traversal
 * satisfy the given Predicate predicate.
 */
public <T> int countSuchAcc(Traversal<T> tr,
                            int acc,
                            Predicate<T> choice){
  if (tr.isEmpty())
    return acc;
  else
    return countSuchAcc(tr.getRest(),
                     updateCountSuch(tr.getFirst(), acc, choice),
                     choice);
}

/** A helper to produce the updated value of the accumulator
 * @param <T> the type of data in this data set
 * @param t the instance of the data set to be used in the update
 * @param acc the current value of the accumulator
 * @param choice the given Predicate predicate.
 * @return the updated value of the accumulator.
 */
protected <T> int updateCountSuch(T t,
                                  int acc,
                                  Predicate<T> choice){
  if (choice.select(t))
    return acc + 1;
  else
    return acc;
}
```

```
/*-*********************************************************************
TEMPLATE-ANALYSIS:
return-type method-name(Traversal tr){
  return-type acc = BASE-VALUE;
  while (CONTINUATION-PREDICATE){
    acc = UPDATE (CURREENT, acc);
    tr  = ADVANCE;
  }
  return acc;
}

COMPLETE METHOD TEMPLATE:
-------------------------
<T> return-type method-name(Traversal<T> tr){
  +------------------------------+
  | return-type acc = BASE-VALUE |;
  +------------------------------+
         +---------------+
  while (| !tr.isEmpty() |)
         +---------------+
  {
          +----------------------------+
    acc = | update(tr.getFirst(), acc) |;
          +----------------------------+
         +--------------+
    tr = | tr.getRest() |;
         +--------------+
  }
  return acc;
}

<T> return-type update(T t, return-type acc){
...
}
*******************************************************************-*/

/** VERSION THAT USES THE while LOOP.
 * Count how many data elements generated by the given traversal
 * satisfy the given Predicate predicate.
 */
 // orMap with while loop and iterator
 public <T> boolean orMapWhile(Traversal<T> tr,
                               Predicate<T> choice){

  // preamble: Define accumulator, initialize it to the BASE-VALUE
  boolean acc = false;

  // loop header: while(continuation-predicate)
  while(!tr.isEmpty()){

    // loop body: update
    acc = updateOrMap(tr.getFirst(), acc, choice);

    // loop advance:
    tr = tr.getRest();
  }

  // postmortem: produce the result
  return acc;
}
```

```
/*-*********************************************************************
TEMPLATE-ANALYSIS:
return-type method-name(Traversal tr){
  return-type acc = BASE-VALUE;
  for (return-type acc = BASE-VALUE; *** DO NOT INCLUDE - DONE ALREADY ***
       CONTINUATION-PREDICATE;
       tr  = ADVANCE){
    acc = UPDATE (CURREENT, acc);
  }
  return acc;
}

COMPLETE METHOD TEMPLATE:
-------------------------
<T> return-type method-name(Traversal<T> tr){
  +------------------------------+
  | return-type acc = BASE-VALUE |;
  +------------------------------+

  for (... no initialization is needed ...;
       +---------------+
       | !tr.isEmpty() |;
       +---------------+
            +--------------+
       tr = | tr.getRest() |)
            +--------------+
  {
           +----------------------------+
     acc = | update(tr.getFirst(), acc) |;
           +----------------------------+
  }
  return acc;
}

<T> return-type update(T t, return-type acc){
...
}
*******************************************************************-*/

 /** IMPERATIVE VERSION THAT USES for LOOP WITH THE Traversal.
  * Count how many data elements generated by the given traversal
  * satisfy the given Predicate predicate.
  */
 // orMap with for loop and iterator
 public <T> boolean orMapFor(Traversal<T> tr, Predicate<T> choice){

  // Define the accumulator and initialize it to the BASE-VALUE;
  boolean acc = false;
  // loop header:
  // for(... accumulator is already defined and initialized ... ;
  //     continuation-predicate;
  //     update)
  for(;
      !tr.isEmpty();
      tr = tr.getRest()){

    // loop body: uses current element
    acc = updateOrMap(tr.getFirst(), acc, choice);
  }

  // postmortem: produce the result
  return acc;
}
```

```
/*-*************************************************************************
TEMPLATE-ANALYSIS:
return-type method-name(ArrayList<T> alist){
  int index; // to represent the traversal
  return-type acc = BASE-VALUE;
  for (index = 0;  // start the traversal at the beginning
       CONTINUATION-PREDICATE;
       index  = ADVANCE){
    acc = UPDATE (CURREENT, acc);
  }
  return acc;
}

COMPLETE METHOD TEMPLATE:
-------------------------
<T> return-type method-name(ArrayList<T> alist){
  +-----------------------------+
  | return-type acc = BASE-VALUE |;
  +-----------------------------+

  for (index = 0;
       +----------------------+
       | index < alist.size() |;
       +----------------------+
               +-----------+
       index = | index + 1 |)
               +-----------+
  {
           +-------------------------------+
     acc = | update(alist.get(index), acc) |;
           +-------------------------------+
  }
  return acc;
}

<T> return-type update(T t, return-type acc){
...
}
```

```
/** IMPERATIVE VERSION THAT USES for LOOP WITH index based traversal.
 * Count how many data elements in thre given ArrayList
 * satisfy the given Predicate predicate.
 */
public <T> boolean orMapForCounted(ArrayList<T> alist,
                                   Predicate<T> choice){

  // Define the accumulator and initialize it to the BASE-VALUE;
  boolean acc = false;
  // loop header:
  // for(... accumulator is already defined and initialized ...
  //     ...BUT initalize the loop index: int index = 0;
  // continuation-predicate: index < alist.size()
  // update: index = index + 1
  for(int index = 0; index < alist.size(); index = index + 1){

    // loop body: uses current element
    acc = updateOrMap(alist.get(index), acc, choice);
  }

  // postmortem: produce the result
  return acc;
}
```