Variables and Constraints

- A constrained integer variable is called an IntVar
- It has a domain of values
- It has a list of constraints
  - The variable is in the scope of those constraints

```
public class IntVar extends Var implements Iterable<Integer> {

    public String name;
    public ArrayList<Constraint> constraints;
    public LinkedList<Integer> deletions;
    Problem pb;
    public Domain domain;

    public IntVar(String name,int lwb, int upb,Problem pb){
        this.name   = name;
        this.pb     = pb;
        domain      = new Domain(pb,lwb,upb);
        constraints = new ArrayList<Constraint>();
        deletions   = new LinkedList<Integer>();
        pb.variables.add(this);
    }
```

- size()
- contains(x)
- remove(x)
- instantiate(x)
- min()
- max()
- removeAbove(x)
- removeBelow(x)
- getValue()
- getLwb()
- getUpb()

```
public int size(){return domain.size();}

public boolean contains(int x){return domain.contains(x);}

public boolean remove(int x){
    boolean changed = domain.remove(x);
    if (changed && pb.propagationOn) changeRemoval(x);
    return changed;
}

public boolean instantiate(int x){
    boolean changed = domain.removeAllBut(x);
    if (changed && pb.propagationOn) changeInstantiate();
    return changed;
}

public boolean isInstantiated(){return domain.size() == 1;}

public int min(){return domain.min();}

public int max(){return domain.max();}

public boolean removeAbove(int x){
    boolean changed = domain.removeAbove(x);
    if (changed && pb.propagationOn) changeUPB();
    return changed;
}

public boolean removeBelow(int x){
    boolean changed = domain.removeBelow(x);
    if (changed && pb.propagationOn) changeLWB();
    return changed;
}
```

The only exception in IntVar

```
public int getValue(){
    if (!isInstantiated()) throw new IntVarException("Not instantiated");
    return domain.getLastValue();
}
```

```
public int size(){return domain.size();}

public boolean contains(int x){return domain.contains(x);}

public boolean remove(int x){
    boolean changed = domain.remove(x);
    if (changed && pb.propagationOn) changeRemoval(x);
    return changed;
}

public boolean instantiate(int x){
    boolean changed = domain.removeAllBut(x);
    if (changed && pb.propagationOn) changeInstantiate();
    return changed;
}

public boolean isInstantiated(){return domain.size() == 1;}

public int min(){return domain.min();}

public int max(){return domain.max();}

public boolean removeAbove(int x){
    boolean changed = domain.removeAbove(x);
    if (changed && pb.propagationOn) changeUPB();
    return changed;
}

public boolean removeBelow(int x){
    boolean changed = domain.removeBelow(x);
    if (changed && pb.propagationOn) changeLWB();
    return changed;
}
```

change removal

```
void changeRemoval(int value){
    if (size() == 1) changeInstantiate();
    if (value > max()) changeUPB();
    if (value < min()) changeLWB();
    if (min() < value && value < max()){
        deletions.add(value);
        for (Constraint c : constraints){
            if (!c.onQueue){
                pb.revisionQueue.add(c);
                c.onQueue = true;
            }
            if (!c.removals.contains(this)) c.removals.add(this);
        }
    }
}
```

change instantiate

```
void changeInstantiate(){
    for (Constraint c : constraints){
        if (!c.onQueue){
            pb.revisionQueue.add(c);
            c.onQueue = true;
        }
        if (!c.instantiations.contains(this)) c.instantiations.add(this);
    }
}
```

Do we need to describe constraints before variables or variables before constraints?

A quick intro to constraints

```
public abstract class Constraint {

    public Problem pb;
    public ArrayList<IntVar> variables;
    public LinkedList<IntVar> removals, newUPB, newLWB, instantiations;
    public boolean onQueue, init, isReified, posted, isOpposite;
    public Constraint opposite;
    public IntVar b; // refication boolean variable
    String name;


    public Constraint(Problem pb){
        this.pb         = pb;
        variables       = new ArrayList<IntVar>();
        removals        = new LinkedList<IntVar>();
        newUPB          = new LinkedList<IntVar>();
        newLWB          = new LinkedList<IntVar>();
        instantiations  = new LinkedList<IntVar>();
        onQueue         = false;
        init            = true;
        isReified       = false;
        posted          = false;
        opposite        = null;
        b               = null;
    }
```

```
public abstract void propagate();

public abstract void propagateInit();

public abstract void propagateRemoval(IntVar x, int value);

public abstract void propagateUPB(IntVar x);

public abstract void propagateLWB(IntVar x);

public abstract void propagateInstantiate(IntVar x);

public abstract boolean isEntailed();

public abstract Constraint makeOpposite();

public abstract boolean isLegal();
```

Revising a constraint
That is, doing all necessary propagations

```
public boolean revise()
    if (isReified){
        try {reifiedPropagation();}
        catch (CPException e) {return false;}
        return true;
    }
    if (init){
        try {propagateInit();}
        catch (CPException e) {return false;}
        init = false;
    }
    while (removals.size() > 0 || newUPB.size() > 0 || newLWB.size() > 0 ||  instantiations.size() > 0){
        while (!removals.isEmpty()){
            IntVar v = removals.remove();
            while (!v.deletions.isEmpty()){
                int x = v.deletions.remove();
                try {propagateRemoval(v,x);}
                catch (CPException e) {return false;}
            }
        }
        while (!newUPB.isEmpty()){
            IntVar v = newUPB.remove();
            try {propagateUPB(v);}
            catch (CPException e) {return false;}
        }
        while (!newLWB.isEmpty()){
            IntVar v = newLWB.remove();
            try {propagateLWB(v);}
            catch (CPException e) {return false;}
        }
        while (!instantiations.isEmpty()){
            IntVar v = instantiations.remove
            try {propagateInstantiate(v);}
            catch (CPException e) {return false;}
        }
    }
    return true;
}
```

```
public boolean revise()
    if (isReified){
        try {reifiedPropagation();}
        catch (CPException e) {return false;}
        return true;
    }
    if (init){
        try {propagateInit();}
        catch (CPException e) {return false;}
        init = false;
    }
    while (removals.size() > 0 || newUPB.size() > 0 || newLWB.size() > 0 ||  instantiations.size() > 0){
        while (!removals.isEmpty()){
            IntVar v = removals.remove();
            while (!v.deletions.isEmpty()){
                int x = v.deletions.remove();
                try {propagateRemoval(v,x);}
                catch (CPException e) {return false;}
            }
        }
        while (!newUPB.isEmpty()){
            IntVar v = newUPB.remove();
            try {propagateUPB(v);}
            catch (CPException e) {return false;}
        }
        while (!newLWB.isEmpty()){
            IntVar v = newLWB.remove();
            try {propagateLWB(v);}
            catch (CPException e) {return false;}
        }
        while (!instantiations.isEmpty()){
            IntVar v = instantiations.remove
            try {propagateInstantiate(v);}
            catch (CPException e) {return false;}
        }
    }
    return true;
}
```

Ignore for time being (the next 2 years?)

```
public boolean revise()
    if (isReified){
        try {reifiedPropagation();}
        catch (CPException e) {return false;}
        return true;
    }
    if (init){
        try {propagateInit();}
        catch (CPException e) {return false;}
        init = false;
    }
    while (removals.size() > 0 || newUPB.size() > 0 || newLWB.size() > 0 ||  instantiations.size() > 0){
        while (!removals.isEmpty()){
            IntVar v = removals.remove();
            while (!v.deletions.isEmpty()){
                int x = v.deletions.remove();
                try {propagateRemoval(v,x);}
                catch (CPException e) {return false;}
            }
        }
        while (!newUPB.isEmpty()){
            IntVar v = newUPB.remove();
            try {propagateUPB(v);}
            catch (CPException e) {return false;}
        }
        while (!newLWB.isEmpty()){
            IntVar v = newLWB.remove();
            try {propagateLWB(v);}
            catch (CPException e) {return false;}
        }
        while (!instantiations.isEmpty()){
            IntVar v = instantiations.remove
            try {propagateInstantiate(v);}
            catch (CPException e) {return false;}
        }
    }
    return true;
}
```

First time revised?

```
public boolean revise()
    if (isReified){
        try {reifiedPropagation();}
        catch (CPException e) {return false;}
        return true;
    }
    if (init){
        try {propagateInit();}
        catch (CPException e) {return false;}
        init = false;
    }
    while (removals.size() > 0 || newUPB.size() > 0 || newLWB.size() > 0 ||  instantiations.size() > 0){
        while (!removals.isEmpty()){
            IntVar v = removals.remove();
            while (!v.delet
                int x = v.d
                try {propagateRemoval(v,x);}
                catch (CPException e) {return false;}
            }
        }
        while (!newUPB.isEmpty()){
            IntVar v = newUPB.remove();
            try {propagateUPB(v);}
            catch (CPException e) {return false;}
        }
        while (!newLWB.isEmpty()){
            IntVar v = newLWB.remove();
            try {propagateLWB(v);}
            catch (CPException e) {return false;}
        }
        while (!instantiations.isEmpty()){
            IntVar v = instantiations.remove
            try {propagateInstantiate(v);}
            catch (CPException e) {return false;}
        }
    }
    return true;
}
```

While the queues of changes on this constraint are not empty

```
public boolean revise()
    if (isReified){
        try {reifiedPropagation();}
        catch (CPException e) {return false;}
        return true;
    }
    if (init){
        try {propagateInit();}
        catch (CPException e) {return false;}
        init = false;
    }
    while (removals.size() > 0 || newUPB.size() > 0 || newLWB.size() > 0 ||  instantiations.size() > 0){
        while (!removals.isEmpty()){
            IntVar v = removals.remove();
            while (!v.deletions.isEmpty()){
                int x = v.deletions.remove();
                try {propagateRemoval(v,x);}
                catch (CPException e) {return false;}
            }
        }
        while (!newUPB.isEmpty()){
            IntVar v = newUPB.remove();
            try {propagateUPB(v);}
            catch (CPException e) {return false;}
```

Propagate all removals from the domains of variables in the scope of this constraint

```
            IntVar v = newLWB.remove();
            try {propagateLWB(v);}
            catch (CPException e) {return false;}
        }
        while (!instantiations.isEmpty()){
            IntVar v = instantiations.remove
            try {propagateInstantiate(v);}
            catch (CPException e) {return false;}
        }
    }
    return true;
}
```

Propagate all removals from the domains of variables in the scope of this constraint

```
while (!removals.isEmpty()){
    IntVar v = removals.remove();
    while (!v.deletions.isEmpty()){
        int x = v.deletions.remove();
        try {propagateRemoval(v,x);}
        catch (CPException e) {return false;}
    }
}
```

Propagate all removals from the domains of variables in the scope of this constraint

```
while (!removals.isEmpty()){
    IntVar v = removals.remove();
    while (!v.deletions.isEmpty()){
        int x = v.deletions.remove();
        try {propagateRemoval(v,x);}
        catch (CPException e) {return false;}
    }
}
```

Remove a variable v from the queue of variables on this constraint that have lost values

Propagate all removals from the domains of variables in the scope of this constraint

```
while (!removals.isEmpty()){
    IntVar v = removals.remove();
    while (!v.deletions.isEmpty()){
        int x = v.deletions.remove();
        try {propagateRemoval(v,x);}
        catch (CPException e) {return false;}
    }
}
```

Propagate the consequences of the deletions of values from v wrt this constraint

Propagate all ***decreases in upper bounds*** of domains of variables in the scope of this constraint

```
while (!newUPB.isEmpty()){
    IntVar v = newUPB.remove();
    try {propagateUPB(v);}
    catch (CPException e) {return false;}
}
```

Propagate all ***increases in lower bounds*** of domains of variables in the scope of this constraint

```
while (!newLWB.isEmpty()){
    IntVar v = newLWB.remove();
    try {propagateLWB(v);}
    catch (CPException e) {return false;}
}
```

Propagate all ***instantiatons*** of variables in the scope of this constraint

```
while (!instantiations.isEmpty()){
    IntVar v = instantiations.remove();
    try {propagateInstantiate(v);}
    catch (CPException e) {return false;}
}
```

… and the Problem holds it all together, thus …

… and the Problem holds it all together, thus …

```
public boolean propagate(){
    boolean consistent = true;
    while (consistent && !revisionQueue.isEmpty()){
        Constraint c = revisionQueue.peek();
        consistent = c.revise();
        if (consistent) {
            revisionQueue.remove();
            c.onQueue = false;
            assert c.isLegal() : c;
        }
        if (!consistent) flushRevisionQueue();
    }
    assert cleanConstraints() : "Dirty constraint queue";
    return consistent;
}
```

Therefore, it comes together like this …

Therefore, it comes together like this …

1. A variable's domain is changed
2. The constraints of that variable are added to the revision queue
3. A constraint is taken from the revision queue
4. The constraint is revised
5. This may reduce the domains of variables in that constraint
6. This causes more constraints to be added to the revision queue
7. If the revision queue is not empty go back to step 3.

Therefore, it comes together like this …

1. A variable's domain is changed
2. The constraints of that variable are added to the revision queue
3. A constraint is taken from the revision queue
4. The constraint is revised
5. This may reduce the domains of variables in that constraint
6. This causes more constraints to be added to the revision queue
7. If the revision queue is not empty go back to step 3.

This terminates because domains can only stay the same or reduce in size.

If all domains are non-empty and no change takes place the queue empties and we are done.

If any domain becomes empty , at step 5, we are done and in a bad world.

This might be thought of as some kind of ***chaotic iteration***

And now, an actual constraint

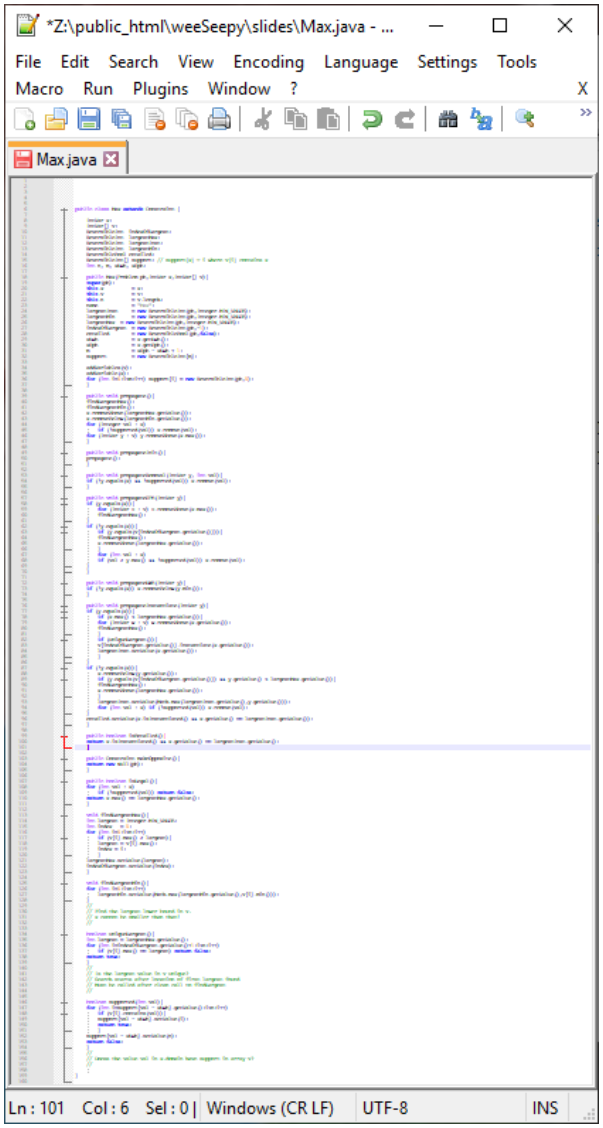
The NotEquals constraint

```
public class NotEquals extends BinaryConstraint {

    public NotEquals(Problem pb,IntVar x, IntVar y){
        super(pb,x,y);
        name = "NotEquals";
        assert !x.equals(y) : "x and y must be different "+ this;
    }

    public void propagate(){
        if (x.isInstantiated()) y.remove(x.getValue());
        if (y.isInstantiated()) x.remove(y.getValue());
    }

    public void propagateInit(){propagate();}

    public void propagateRemoval(IntVar v, int value){}

    public void propagateUPB(IntVar v){}

    public void propagateLWB(IntVar v){}

    public void propagateInstantiate(IntVar v){propagate();}

    public boolean isEntailed(){
        return x.max() < y.min() || x.min() > y.max();
    }
```

A summation constraint

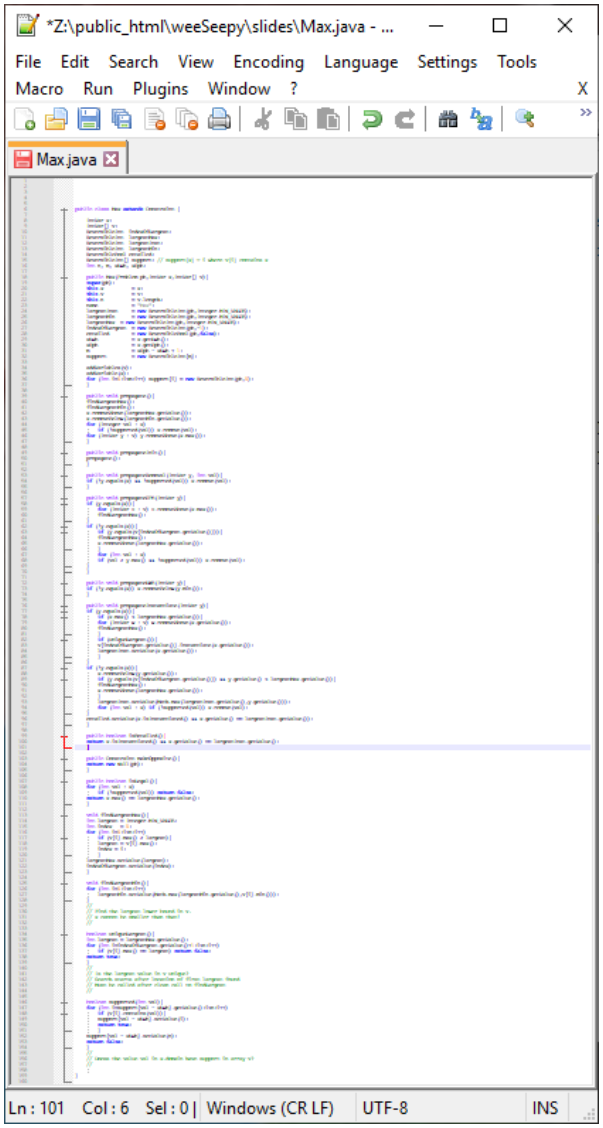
```
public class SumGEQ extends Constraint {

    int c, n;
    int[] a;
    IntVar[] x;

    public SumGEQ(Problem pb, int[] a, IntVar[] x, int c){
        super(pb);
        this.c = c;
        this.a = a;
        this.x = x;
        this.n = x.length;
        name   = "SumGEQ (with c = "+ c +"): ";
        addVariables(x);
        assert allPositive() : "all coeffs must be > 0 "+ this;
    }

    public void propagate(){
        int sumOfMaxs = 0;
        for (int i=0;i<n;i++) sumOfMaxs = sumOfMaxs + a[i] * x[i].max();
        int slack = c - sumOfMaxs;
        for (int j=0;j<n;j++){
            int S_j = slack + a[j] * x[j].max(); // max slack on a[j]*x[j].max()
            x[j].removeBelow(S_j/a[j]); // can have no less than this!
        }
    }

    public void propagateInit(){propagate();}

    public void propagateRemoval(IntVar x, int value){}

    public void propagateUPB(IntVar x){propagate();}

    public void propagateLWB(IntVar x){propagate();}

    public void propagateInstantiate(IntVar x){propagate();}
```

The Max constraint



The Max constraint

About 150 lines of code



… and now we should look quickly at search
(now that everything is in place)