

```

public class PredsValidator {

    /**
     * Validates cyclic references among <code>node</code> and its
     * predecessors.
     *
     * @param node
     */
    public static void validatePredecessors(final ScheduleNode node) {
        if (node.getPredecessors().size() == 0) {
            // No preds to validate
            return;
        }

        for (ScheduleNode pred : node.getPredecessors()) {
            // Start validations given the current pred
            if (isSelf(node, pred)) {
                // Handle cyclic ref.
            }
            if (isChild(node, pred)) {
                // Handle cyclic ref.
            }
            if (isParent(node, pred)) {
                // Handle cyclic ref.
            }
            if (isSuccessorOfChild(node, pred)) {
                // Handle cyclic ref.
            }
            if (isSuccessorOfParent(node, pred)) {
                // Handle cyclic ref.
            }
            if (isSuccessor(node, pred)) {
                // Handle cyclic ref.
            }
        }
    }

    /**
     * @param curNode
     * @param subject
     * @return <code>true</code> if subject is successor of <code>curNode</code>
     *         , <code>false</code> otherwise.
     */
    private static boolean isSuccessor(final ScheduleNode curNode,
                                       final ScheduleNode subject) {
        for (ScheduleNode successor : curNode.getSuccessors()) {
            if (subject.getId().equals(successor.getId())) {
                // Subject is successor of node
                return true;
            }
            if (isParent(successor, subject)) {
                // Subject is parent of successor
                return true;
            }
            if (isChild(successor, subject)) {
                // Subject is child of successor
                return true;
            }
            if (isSuccessor(successor, subject)) {

```

PredsValidator.java

```

        // Subject is successor of successor
        return true;
    }
}

return false;
}

/**
 * @param curNode
 * @param subject
 * @return <code>true</code> if <code>subject</code> is successor of
 *         <code>curNode</code> 's parent, <code>false</code> otherwise.
 */
private static boolean isSuccessorOfParent(final ScheduleNode curNode,
    final ScheduleNode subject) {
    final ScheduleNode parent = (ScheduleNode) curNode.getParent();
    if (parent == null) {
        // A parent was not found with the same subject id
        return false;
    }
    return isSuccessor(parent, subject);
}

/**
 * @param curNode
 * @param subject
 * @return <code>true</code> if <code>subject</code> is successor of one of
 *         <code>curNode</code>'s children, <code>false</code> otherwise.
 */
private static boolean isSuccessorOfChild(final ScheduleNode curNode,
    final ScheduleNode subject) {
    // Subject is not child, but it could be successor of child
    for (TreeNode child : curNode.getChildren()) {
        if (isSuccessor((ScheduleNode) child, subject)) {
            return true;
        }
    }

    // Subject is not successor of child
    return false;
}

/**
 * @param curNode
 * @param subject
 * @return <code>true</code> if <code>subject</code> is parent of
 *         <code>curNode</code>, <code>false</code> otherwise.
 */
private static boolean isParent(final ScheduleNode curNode,
    final ScheduleNode subject) {
    final ScheduleNode parent = (ScheduleNode) curNode.getParent();
    if (parent == null) {
        // A parent was not found with the same subject id
        return false;
    }

    if (subject.getId().equals(parent.getId())) {
        // Subject is same as parent of current node
        return true;
    }
}

```

```

    }

    if (isSuccessor(parent, subject)) {
        // Subject is successor of parent
        return true;
    }

    // Determine if subject is parent of parent
    return isParent(parent, subject);
}

/**
 * @param curNode
 * @param subject
 * @return <code>>true</code> if <code>subject</code> is the same node as
 *         curNode, <code>>false</code> otherwise.
 */
private static boolean isSelf(final ScheduleNode curNode,
    final ScheduleNode subject) {
    if (subject.getIndx().equals(curNode.getIndx())) {
        return true;
    }
    return false;
}

/**
 * @param curNode
 * @param subject
 * @return <code>true</code> if <code>subject</code> is child of
 *         <code>curNode</code>, <code>>false</code> otherwise.
 */
private static boolean isChild(final ScheduleNode curNode,
    final ScheduleNode subject) {
    final Long min = curNode.getIndx();
    final Long max = min + curNode.getAbsChildCount();
    if (subject.getIndx() > min && subject.getIndx() <= max) {
        // Subject is child of node
        return true;
    }

    for (TreeNode child : curNode.getChildren()) {
        if (isSuccessor((ScheduleNode) child, subject)) {
            // Subject is successor of child of node
            return true;
        }
    }

    return false;
}
}
}

```