Before regression:
svn checkout -r1573075 https://svn.apache.org/repos/asf/jackrabbit/oak/trunk
/oak-core/src/main/java/org/apache/jackrabbit/oak/core/MutableTree.java

```
@Override
public boolean remove() {
        beforeWrite();
        if (parent != null && parent.hasChild(name)) {
                nodeBuilder.remove();
                if (parent.hasOrderableChildren()) {
                        parent.nodeBuilder.setProperty(
                        PropertyBuilder.copy(NAME,
                           parent.nodeBuilder.getProperty(TreeConstants.OAK_CHILD_ORDER))
                        .removeValue(name)
                        .getPropertyState()
                        );
                }
                root.updated();
                return true;
        } else {
        return false;
}
}

@Override
public Tree addChild(String name) {
        checkArgument(!isHidden(name));
        beforeWrite();
        if (!super.hasChild(name)) {
                nodeBuilder.setChildNode(name);
                if (hasOrderableChildren()) {
                        nodeBuilder.setProperty(
                        PropertyBuilder.copy(NAME,
                           nodeBuilder.getProperty(TreeConstants.OAK_CHILD_ORDER))
                        .addValue(name)
                        .getPropertyState());
                }
                root.updated();
        }
        return createChild(name);
}

@Override
public void setOrderableChildren(boolean enable) {
        beforeWrite();
        if (enable) {
                ensureChildOrderProperty();
        } else {
        nodeBuilder.removeProperty(TreeConstants.OAK_CHILD_ORDER);
}
}

@Override
public boolean orderBefore(final String name) {
        beforeWrite();
        if (parent == null) {
                // root does not have siblings
                return false;
        }
        if (name != null) {
```

```
                if (name.equals(this.name) || !parent.hasChild(name)) {
                        // same node or no such sibling (not existing or not accessible)
                        return false;
                }
        }
        // perform the reorder
        parent.ensureChildOrderProperty();
        // all siblings but not this one
        Iterable<String> siblings = filter(
        parent.getChildNames(),
        new Predicate<String>() {
                @Override
                public boolean apply(String name) {
                        return !MutableTree.this.name.equals(name);
                }
        });
        // create head and tail
        Iterable<String> head;
        Iterable<String> tail;
        if (name == null) {
                head = siblings;
                tail = Collections.emptyList();
        } else {
        int idx = indexOf(siblings, new Predicate<String>() {
                @Override
                public boolean apply(String sibling) {
                        return name.equals(sibling);
                }
        });
        head = Iterables.limit(siblings, idx);
        tail = Iterables.skip(siblings, idx);
}
// concatenate head, this name and tail
parent.nodeBuilder.setProperty(
MultiGenericPropertyState.nameProperty(
TreeConstants.OAK_CHILD_ORDER, Iterables.concat(head, Collections.singleton(getName()),
    tail))
);
root.updated();
return true;
}



/**
* Update the child order with children that have been removed or added.
* Added children are appended to the end of the {@link
    org.apache.jackrabbit.oak.plugins.tree.TreeConstants#OAK_CHILD_ORDER}
* property.



*/
void updateChildOrder() {
        if (!hasOrderableChildren()) {
                return;
        }
        Set<String> names = Sets.newLinkedHashSet();
        for (String name : getChildNames()) {
                if (nodeBuilder.hasChildNode(name)) {
```

```
                        names.add(name);
                }
        }
        for (String name : nodeBuilder.getChildNodeNames()) {
                names.add(name);
        }
        PropertyBuilder<String> builder = PropertyBuilder.array(NAME,
           TreeConstants.OAK_CHILD_ORDER);
        builder.setValues(names);
        nodeBuilder.setProperty(builder.getPropertyState());
}
/**
* Ensures that the {@link
   org.apache.jackrabbit.oak.plugins.tree.TreeConstants#OAK_CHILD_ORDER} exists. This
   method will create
* the property if it doesn't exist and initialize the value with the names
* of the children as returned by {@link NodeBuilder#getChildNodeNames()}.
*/

private void ensureChildOrderProperty() {
        if (!nodeBuilder.hasProperty(TreeConstants.OAK_CHILD_ORDER)) {
                nodeBuilder.setProperty(
                MultiGenericPropertyState.nameProperty(TreeConstants.OAK_CHILD_ORDER,
                   nodeBuilder.getChildNodeNames()));
        }
}



//failed test:

org.apache.jackrabbit.mk.store.DefaultRevisionStoreTest
org.apache.jackrabbit.mk.MicroKernelImplTest
org.apache.jackrabbit.oak.plugins.document.SimpleTest
org.apache.jackrabbit.oak.security.authorization.accesscontrol.AccessControlManagerImplTest
```

Regressed version:
svn checkout -r1579349 https://svn.apache.org/repos/asf/jackrabbit/oak/trunk
/oak-core/src/main/java/org/apache/jackrabbit/oak/core/MutableTree.java

```
@Override
public boolean remove() {
        beforeWrite();
        if (parent != null && parent.hasChild(name)) {
                nodeBuilder.remove();
                updateChildOrder(false);
                root.updated();
                return true;
        } else {
        return false;
}
}

@Override
public Tree addChild(String name) {
        checkArgument(!isHidden(name));
        beforeWrite();
        if (!super.hasChild(name)) {
                nodeBuilder.setChildNode(name);
                updateChildOrder(false);
                root.updated();
        }
        return createChild(name);
}

@Override
public void setOrderableChildren(boolean enable) {
        beforeWrite();
        if (enable) {
                updateChildOrder(true);
        } else {
        nodeBuilder.removeProperty(OAK_CHILD_ORDER);
}
}

@Override
public boolean orderBefore(final String name) {
        beforeWrite();
        if (parent == null) {
                // root does not have siblings
                return false;
        }
        if (name != null) {
                if (name.equals(this.name) || !parent.hasChild(name)) {
                        // same node or no such sibling (not existing or not accessible)
                        return false;
                }
        }
        // perform the reorder
        List<String> names = newArrayList();

        for (String n : parent.getChildNames()) {
                if (n.equals(name)) {
                        names.add(this.name);

                }
```

```
                if (!n.equals(this.name)) {
                        names.add(n);
                }

        }
        if (name == null) {
                names.add(this.name);

        }
        parent.nodeBuilder.setProperty(OAK_CHILD_ORDER, names, NAMES);
        root.updated();
        return true;
}

/**
* Updates the child order to match any added or removed child nodes that
* are not yet reflected in the {@link TreeConstants#OAK_CHILD_ORDER}
* property. If the {@code force} flag is set, the child order is set
* in any case, otherwise only if the node already is orderable.
*
* @param force whether to add child order information if it doesn't exist
*/
void updateChildOrder(boolean force) {
        if (force || hasOrderableChildren()) {
                nodeBuilder.setProperty(PropertyStates.createProperty(


                OAK_CHILD_ORDER, getChildNames(), Type.NAMES));

        }

}


//failed test:

org.apache.jackrabbit.mk.store.DefaultRevisionStoreTest
org.apache.jackrabbit.mk.MicroKernelImplTest
org.apache.jackrabbit.oak.plugins.document.SimpleTest
org.apache.jackrabbit.oak.security.authorization.accesscontrol.AccessControlManagerImplTest

org.apache.jackrabbit.oak.plugins.document.blob.RDBBlobStoreTest extends
    AbstractBlobStoreTest

// failed test in AbstractBlobStoreTest:
   @Test
   public void delete() throws Exception {
        Set<String> ids = createArtifacts();

        store.deleteChunks(Lists.newArrayList(ids), 0);

        Iterator<String> iter = store.getAllChunkIds(0);
        Set<String> ret = Sets.newHashSet();
        while (iter.hasNext()) {
                ret.add(iter.next());
        }

        assertTrue(ret.toString(), ret.isEmpty());
   }
```

Corrected version:
svn checkout -r1579637 https://svn.apache.org/repos/asf/jackrabbit/oak/trunk
/oak-core/src/main/java/org/apache/jackrabbit/oak/core/MutableTree.java

```
import static com.google.common.collect.Sets.newLinkedHashSet;
import java.util.Set;


@Override
public boolean remove() {
        beforeWrite();
        if (parent != null && parent.hasChild(name)) {
                nodeBuilder.remove();
                PropertyState order = parent.nodeBuilder.getProperty(OAK_CHILD_ORDER);
                if (order != null) {
                        Set<String> names = newLinkedHashSet(order.getValue(NAMES));
                        names.remove(name);
                        parent.nodeBuilder.setProperty(OAK_CHILD_ORDER, names, NAMES);
                }
                root.updated();
                return true;
        } else {
        return false;
}
}

@Override
public Tree addChild(String name) {
        checkArgument(!isHidden(name));
        beforeWrite();
        if (!super.hasChild(name)) {
                nodeBuilder.setChildNode(name);
                PropertyState order = nodeBuilder.getProperty(OAK_CHILD_ORDER);
                if (order != null) {
                        Set<String> names = newLinkedHashSet(order.getValue(NAMES));
                        names.add(name);
                        nodeBuilder.setProperty(OAK_CHILD_ORDER, names, NAMES);
                }
                root.updated();
        }
        return createChild(name);
}



//failed test:

org.apache.jackrabbit.mk.store.DefaultRevisionStoreTest
org.apache.jackrabbit.mk.MicroKernelImplTest
org.apache.jackrabbit.oak.plugins.document.SimpleTest
org.apache.jackrabbit.oak.security.authorization.accesscontrol.AccessControlManagerImplTest
```