Item 181
git_comments:

1. if namedArgs.containsKey(propertyName) setProperty(propertyName, namedArgs.get(propertyName));
2. add map constructor if needed, don't do it for LinkedHashMap for now (would lead to duplicate signature) or if there is only one Map property (for backwards compatibility)
3. add a no-arg constructor too
4. GROOVY-5243: special support for Map, Object, AbstractMap, HashMap but currently not LinkedHashMap

git_commits:

1. **summary:** GROOVY-5243: @Canonical @TupleConstructor can't handle Object or Map properties
   message: GROOVY-5243: @Canonical @TupleConstructor can't handle Object or Map properties

github_issues:

github_issues_comments:

github_pulls:

github_pulls_comments:

github_pulls_reviews:

jira_issues:

1. **summary:** @Canonical @TupleConstructor can't handle Object or Map properties
   description: Given the following code, I would expect the assertions to pass. They currently don't. However, if the {{a}} and {{b}} properties are typed to String, for example, everything goes as expected. It might be challenging to get this to work properly for more complex types, but it would make the annotations *really* transparent. The way it currently behaves is very unsettling (a is set with a Map containing both a and b with their respective values, while b is not set to anything) {code} import groovy.transform.Canonical def weird = new CanonicalIsWeird(a: 'first letter', b: 'second letter') println "a: ${weird.a}" println "b: ${weird.b}" assert weird.a == 'first letter' assert weird.b == 'second letter' @Canonical class CanonicalIsWeird { def a, b } {code} https://gist.github.com/1565938
   label: test
2. **summary:** @Canonical @TupleConstructor can't handle Object or Map properties
   description: Given the following code, I would expect the assertions to pass. They currently don't. However, if the {{a}} and {{b}} properties are typed to String, for example, everything goes as expected. It might be challenging to get this to work properly for more complex types, but it would make the annotations *really* transparent. The way it currently behaves is very unsettling (a is set with a Map containing both a and b with their respective values, while b is not set to anything) {code} import groovy.transform.Canonical def weird = new CanonicalIsWeird(a: 'first letter', b: 'second letter') println "a: ${weird.a}" println "b: ${weird.b}" assert weird.a == 'first letter' assert weird.b == 'second letter' @Canonical class CanonicalIsWeird { def a, b } {code} https://gist.github.com/1565938
3. **summary:** @Canonical @TupleConstructor can't handle Object or Map properties
   description: Given the following code, I would expect the assertions to pass. They currently don't. However, if the {{a}} and {{b}} properties are typed to String, for example, everything goes as expected. It might be challenging to get this to work properly for more complex types, but it would make the annotations *really* transparent. The way it currently behaves is very unsettling (a is set with a Map containing both a and b with their respective values, while b is not set to anything) {code} import groovy.transform.Canonical def weird = new CanonicalIsWeird(a: 'first letter', b: 'second letter') println "a: ${weird.a}" println "b: ${weird.b}" assert weird.a == 'first letter' assert weird.b == 'second letter' @Canonical class CanonicalIsWeird { def a, b } {code} https://gist.github.com/1565938
4. **summary:** @Canonical @TupleConstructor can't handle Object or Map properties
   description: Given the following code, I would expect the assertions to pass. They currently don't. However, if the {{a}} and {{b}} properties are typed to String, for example, everything goes as expected. It might be challenging to get this to work properly for more complex types, but it would make the

annotations *really* transparent. The way it currently behaves is very unsettling (a is set with a Map containing both a and b with their respective values, while b is not set to anything) {code} import groovy.transform.Canonical def weird = new CanonicalIsWeird(a: 'first letter', b: 'second letter') println "a: ${weird.a}" println "b: ${weird.b}" assert weird.a == 'first letter' assert weird.b == 'second letter' @Canonical class CanonicalIsWeird { def a, b } {code} https://gist.github.com/1565938

5. **summary:** @Canonical @TupleConstructor can't handle Object or Map properties
   description: Given the following code, I would expect the assertions to pass. They currently don't. However, if the {{a}} and {{b}} properties are typed to String, for example, everything goes as expected. It might be challenging to get this to work properly for more complex types, but it would make the annotations *really* transparent. The way it currently behaves is very unsettling (a is set with a Map containing both a and b with their respective values, while b is not set to anything) {code} import groovy.transform.Canonical def weird = new CanonicalIsWeird(a: 'first letter', b: 'second letter') println "a: ${weird.a}" println "b: ${weird.b}" assert weird.a == 'first letter' assert weird.b == 'second letter' @Canonical class CanonicalIsWeird { def a, b } {code} https://gist.github.com/1565938

jira_issues_comments:

1. **body:** Unrelated, but I can't seem to be able to edit the issue. Forgot to check the "testcase submitted" box (although said test case is very minimal)
   label: test
2. Just for further explanation, by using @TupleConstructor, the following constructors are generated: {code} CanonicalIsWeird(Object a, Object b) CanonicalIsWeird(Object a) CanonicalIsWeird() {code} Normally when using named parameters (i.e. {{new CanonicalIsWeird(a: 'foo', b:' bar')}}) it is represented by storing the named args in a Map which is then (roughly) translated into calling the no-arg constructor followed by setting the properties - approximately what you see below: {code} def obj = new CanonicalIsWeird(Map args) {code} translated into: {code} def obj = new CanonicalIsWeird() obj.setA('foo') obj.setB('bar') {code} but in your case, the single Object constructor matches the Map and alters the desired behavior.
3. OK, it turns out not to be easy to cover all cases since for some scenarios supporting Groovy's named arguments could be in conflict with a user trying to use Map properties. But the good news is cases like yours can and should be supported. So your example and similar ones should now work unless the first property happens to be a LinkedHashMap or there is only one property of Map, HashMap, AbstractMap or LinkedHashMap. The doco has been updated to reflect this limitation.