```python
#!/usr/bin/env python3
# -*-coding:Utf-8 -*

# The MIT License (MIT)
#
# Copyright (c) 2016 Rémi Blaise <remi.blaise@gmx.fr> "http://php-zzortell.rhcloud.com/"
#
# Permission is hereby granted, free of charge, to any person obtaining a copy
# of this software and associated documentation files (the "Software"), to deal
# in the Software without restriction, including without limitation the rights
# to use, copy, modify, merge, publish, distribute, sublicense, and/or sell
# copies of the Software, and to permit persons to whom the Software is
# furnished to do so, subject to the following conditions:
#
# The above copyright notice and this permission notice shall be included in all
# copies or substantial portions of the Software.
#
# THE SOFTWARE IS PROVIDED "AS IS", WITHOUT WARRANTY OF ANY KIND, EXPRESS OR
# IMPLIED, INCLUDING BUT NOT LIMITED TO THE WARRANTIES OF MERCHANTABILITY,
# FITNESS FOR A PARTICULAR PURPOSE AND NONINFRINGEMENT. IN NO EVENT SHALL THE
# AUTHORS OR COPYRIGHT HOLDERS BE LIABLE FOR ANY CLAIM, DAMAGES OR OTHER
# LIABILITY, WHETHER IN AN ACTION OF CONTRACT, TORT OR OTHERWISE, ARISING FROM,
# OUT OF OR IN CONNECTION WITH THE SOFTWARE OR THE USE OR OTHER DEALINGS IN THE
# SOFTWARE.


from textwrap import indent
from operator import itemgetter, attrgetter


class XMLRepr:
    """
    Awesome XML representation base class

    Inherit to have an XML-like repr of instances.

    __repr__ expects:
        attributes to be a list of attribute names to filter and order.
        __dict__ to be a dict, substitute of self.__dict__
        displayChildrenNames to be a bool.
        displaySequencesNames to be a bool.
        indent_prefix to be a string.

    Features:
        - Use class name as tag name.
        - Use non-XMLRepr non-XMLRepr-containing-sequence attributes as
          attributes: names are used as names and values as values.
        - Use XMLRepr attributes as children, printed as it.
        - Use sequence attributes containing exclusively XMLRepr items as
          children: name is used as tag name and items as children.
        - If attributes is not given, order attributes and children by asc.
        - If displayChildrenNames is set True, children are preceded by their
          attr name. Ex: <brick>: <AwesomeBrick id=0 content='Red mushroom'/>
        - Filter attributes with the attribute names given by attributes parameter.
          Futhermore, it indicates the order of attributes.
        - Substitute self.__dict__ by __dict__.
        - If displaySequencesNames is set False, sequences' children are displayed
          without wrapping.

    Example:
        class MyAwesomeClass(XMLRepr):
            def __init__(self):
                self.color = 'pink'
                self.checked = True
                self.brick = AwesomeBrick(0)
                self.bricks = [AwesomeBrick(1), AwesomeBrick(2)]
        class AwesomeBrick(XMLRepr):
            def __init__(self, id):
                self.content = 'Red mushroom'
                self.id = id

        awesome_object = MyAwesomeClass()
        print(awesome_object)

    Output:
        <MyAwesomeClass color='pink' checked=True>
            <AwesomeBrick id=0 content='Red mushroom'/>
            <bricks>
                <AwesomeBrick id=1 content='Red mushroom'/>
                <AwesomeBrick id=2 content='Red mushroom'/>
            </bricks>
        </MyAwesomeClass>
    """

    def __repr__(self,
            attributes = None, __dict__ = None,
            displayChildrenNames = False, displaySequencesNames = True,
            indent_prefix = '    '
        ):
        if __dict__ is None:
            __dict__ = self.__dict__
        if attributes is None:
            attributes_and_children  = __dict__.items()
        else:
            attributes_and_children  = [(attr, __dict__[attr]) for attr in attributes]
        attributeList  = []
        children  = []
        sequences  = []
        for name, value in attributes_and_children :
            if isinstance(value, XMLRepr):
                if displayChildrenNames:
                    children.append((name, value))
                else:
                    children.append(value)
            elif hasattr(value, '__iter__') and all(isinstance(item, XMLRepr) for item in value):
                sequences.append((name, value))
            else:
                attributeList.append((name, value))

        if attributes is None:
            attributeList.sort(key=itemgetter(0))
            if displayChildrenNames:
                children.sort(key=itemgetter(0))
            else:
                children.sort(key=attrgetter('__class__.__name__'))
            sequences.sort(key=itemgetter(0))

        def formatAttributes(attributeList):
            formatted_attributes = ''
            for name, value in attributeList:
                formatted_attributes += '{}={} '.format(name, repr(value))
            return formatted_attributes.rstrip(' ')

        def formatChildren(children):
            formatted_children = ''
            for value in children:
                formatted_children += '{}\n'.format(repr(value))
            return indent(formatted_children, indent_prefix)

        def formatChildrenWithNames(children):
            formatted_children = ''
            for name, value in children:
                formatted_children += '<{}>: {}\n'.format(name, repr(value))
            return indent(formatted_children, indent_prefix)

        def formatSequences(sequences):
            formatted_sequences = ''
            for name, seq in sequences:
                formatted_sequences += formatChildren(seq)
            return formatted_sequences

        def formatSequencesWithNames(sequences):
            formatted_sequences = ''
            for name, seq in sequences:
                formatted_sequences += '<{0}>\n{1}</{0}>\n'.format(name, formatChildren(seq))
            return indent(formatted_sequences, indent_prefix)

        if children or sequences:
            return '<{0} {1}>\n{2}{3}</{0}>'.format(
                self.__class__.__name__,
                formatAttributes(attributeList),
                formatChildrenWithNames(children) if displayChildrenNames \
                else formatChildren(children),
                formatSequencesWithNames(sequences) if displaySequencesNames \
                else formatSequences(sequences)
            )

        return '<{0} {1}/>'.format(
            self.__class__.__name__,
            formatAttributes(attributeList)
        )


if __name__ == '__main__':
    class MyAwesomeClass(XMLRepr):
        def __init__(self):
            self.color = 'pink'
            self.checked = True
            self.brick = AwesomeBrick(0)
            self.awesome = SuperAwesomeBrick(42)
            self.bricks = [AwesomeBrick(1), AwesomeBrick(2)]
    class AwesomeBrick(XMLRepr):
        def __init__(self, id):
            self.content = 'Red mushroom'
            self.id = id
    class SuperAwesomeBrick(AwesomeBrick):
        pass

    awesome_object = MyAwesomeClass()
    print(69*'-')
    print(awesome_object)

    class DisplayNamesAwesomeClass(MyAwesomeClass):
        def __repr__(self):
            return super().__repr__(displayChildrenNames=True, indent_prefix='    ')
    print(DisplayNamesAwesomeClass())

    class FilterAwesomeClass(MyAwesomeClass):
        def __repr__(self):
            return super().__repr__(attributes=['color', 'bricks'], indent_prefix='\t')
    print(FilterAwesomeClass())

    class SubstituteAwesomeClass(MyAwesomeClass):
        def __repr__(self):
            return super().__repr__(__dict__={'color': 'blood'}, indent_prefix='\t')
    print(SubstituteAwesomeClass())

    class WithoutSequencesNamesAwesomeClass(MyAwesomeClass):
        def __repr__(self):
            return super().__repr__(displaySequencesNames=False)
    print(WithoutSequencesNamesAwesomeClass ())
    print(69*'-')
```