```python
#!/usr/bin/python3.4
# -*-coding:Utf-8 -*

from abc import ABCMeta, abstractmethod


class GeneticElementFactory (metaclass=ABCMeta):
    """Handle the evolution logic of a GeneticElement

    This is an abstract class to inherit.
    This is a static class.
    It brings the evolution logic of the GeneticElement through the following
    class methods:
        +create() -> GeneticElement
        +mutate(GeneticElement)
        +combine(GeneticElement, GeneticElement) -> GeneticElement
        +breed(GeneticElement, GeneticElement) -> GeneticElement
    Evolution logic may typically use recursive process over children of elements.
    """

    @property
    @abstractmethod
    def genetic_element_class (self):
        """The GeneticElement based class """

        raise NotImplementedError


    @staticmethod
    @abstractmethod
    def create(parent = None, children = [], cascade = True):
        """Create a GeneticElement from void

        An essential element of the generation proccess.
        This is a static method which has to be implemented.

        return GeneticElement
        """

        raise NotImplementedError


    @staticmethod
    @abstractmethod
    def mutate(element):
        """Operates a genetic mutation

        This is a static method which has to be implemented.

        This is rather designed for internal use, see generate() instead.
        """

        raise NotImplementedError


    @staticmethod
    def combine(element1, element2):
        """Form a new GeneticElement, combination of two ones

        Combine two GeneticElement to form an offspring.
        This is a static method which has to be implemented.

        This is rather designed for internal use, see generate() instead.

        Expects:
            element1, element2 to be GeneticElement's

        return GeneticElement
        """

        raise NotImplementedError


    @classmethod
    def breed(cls, element1, element2):
        """Generate a new GeneticElement, final offspring of two ones

        Call combine() then mutate().
        This is a class method.

        Expects:
            element1, element2 to be a GeneticElement's

        return GeneticElement
        """

        new_element  = cls.combine(element1, element2)
        cls.mutate(new_element)
        return new_element
```