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

from abc import ABCMeta, abstractmethod


class AbstractLogger(metaclass=ABCMeta):
    """Log Generator events

    An abstract logger to implement, by defining the write() and overwrite() methods.
    """

    @abstractmethod
    def write(self, msg):
        """Write a message

        To implement. Do not forget to add a newline ;)
        """

        raise NotImplementedError()


    def overwrite(self, msg):
        """Overwrite the preceding message

        Usefull for interactive shells.
        By default, use write(). To implement.
        """

        self.write(msg)


    def drawProgressBar(self, ratio):
        return (
            '['
            + int(ratio * 50) * '-'
            + (int(ratio) < 1) * '>'
            + (50 - int(ratio * 50)) * ' '
            + ']'
        )


    def onProcessusResume(self, event):
        if event.event_name == 'grading.progress':
            self.count_ia = len(event.grading)

    def onProcessusStart(self, event):
        self.write('Processus {} starts!'.format(event.processus_id))
        self.write(
            'Processus parameters: {} populations of {} individuals are doing to be generated. '
            .format(event.generations, event.pop_length)
        )
        self.write(
            'Selection parameters: selects {}% of the population whose {}% are random.'
            .format(self._percent(event.proportion), self._percent(event.chance))
        )

    def onProcessusDone(self, event):
        self.write('Processus {} is done!'.format(event.processus_id))

    def onCreationStart(self, event):
        self.write('- Creates the initial population... ')

    def onCreationDone(self, event):
        self.overwrite('- Initial population created. ')

    def onGenerationStart(self, event):
        self.write('- Starts generation {}:'.format(event.generation_id))

    def onGenerationDone(self, event):
        self.write('    Generation {} is done.'.format(event.generation_id))

    def onSelectionStart(self, event):
        self.write('    Starts selection. ')

    def onSelectionDone(self, event):
        self.write('    Selection done. ')

    def onGradingStart(self, event):
        self.write('    Start grading... ')

        self.count_ia = 0

    def onGradingProgress(self, event):
        self.count_ia += 1

        self.overwrite('    Grading: {} IA {} gets a score of {}.'.format(
            self.drawProgressBar(self.count_ia / event.pop_length),
            event.individual.id, event.graduation.score
        ))

    def onGradingDone(self, event):
        self.overwrite('    Grading done. ')

    def onBreedingStart(self, event):
        self.write('    Starts breeding. ')

        self.count_ia = 0

```

```
100         def onBreedingProgress(self, event):
101             self.count_ia += 1
102 
103             self.overwrite('    Breeding: {} {} + {} -> {}.'.format(
104                 self.drawProgressBar(self.count_ia / event.pop_length),
105                 event.parents[0].id, event.parents[1].id, event.offspring.id
106             ))
107 
108         def onBreedingDone(self, event):
109             self.overwrite('    Breeding done.')
110 
111 
112         def _percent(self, ratio):
113             return int(100 * ratio)
```