```
class DesignatorDescription(ABC):
    """
    :ivar resolve: The specialized_designators function to use for this designator, defaults to self.ground
    """

    def __init__(self, resolver: Optional[Callable] = None):
        """
        Create a Designator description.

        :param resolver: The grounding method used for the description. The grounding method creates a
location instance that matches the description.
        """
        if resolver is None:
            self.resolve = self.ground

    def ground(self) -> Any:
        """
        Should be overwritten with an actual grounding function which infers missing properties.
        """

class LocationDesignatorDescription(DesignatorDescription):
    """
    Parent class of location designator descriptions.
    """

    @dataclass
    class Location:
        """
        Resolved location that represents a specific point in the world
        """
        pose: Pose
        """
        The resolved pose of the location designator. Pose is inherited by all location designator.
        """

    def __init__(self, resolver=None):
        super().__init__(resolver)

    def ground(self) -> Location:
        """
        Find a location that satisfies all constrains.
        """

class CostmapLocation(LocationDesignatorDescription):
    """
    Used to find the locations suitable for the robot to either reach/visible the target
    """

    @dataclasses.dataclass
    class Location(LocationDesignatorDescription.Location):
        reachable_arms: List[str]
        """
        List of arms with which the pose can be reached, is only used when the 'reachable_for' parameter
is used
        """

    def __init__(self, target: Union[Pose, ObjectDesignatorDescription.Object],
                 reachable_for: Optional[ObjectDesignatorDescription.Object] = None,
                 visible_for: Optional[ObjectDesignatorDescription.Object] = None,
                 reachable_arm: Optional[str] = None, resolver: Optional[Callable] = None):
        """
        Location designator that uses costmaps to calculate locations from which the robot can either
reach or see the target object

        :param target: Location for which visibility or reachability should be calculated
        :param reachable_for: Object for which the reachability should be calculated, usually a robot
        :param visible_for: Object for which the visibility should be calculated, usually a robot
        :param reachable_arm: An optional arm with which the target should be reached

        """

    def ground(self) -> Location:
        """
        Default specialized_designators which returns the first result from the iterator of this instance.

        :return: A resolved location
        """

class SemanticCostmapLocation(LocationDesignatorDescription):
    """
    Used to calculate Locations over semantic entities suitable for the robot, like a table surface
    """

    @dataclasses.dataclass
    class Location(LocationDesignatorDescription.Location):
        pass

    def __init__(self, urdf_link_name, part_of, for_object=None, resolver=None):
        """
        Creates a distribution over a urdf link to sample poses which are on this link. Can be used, for
example, to find
        poses that are on a table. Optionally an object can be given for which poses should be calculated,
in that case
        the poses are calculated such that the bottom of the object is on the link.

        :param urdf_link_name: Name of the urdf link for which a distribution should be calculated
        :param part_of: Object of which the urdf link is a part
        :param for_object: Optional object that should be placed at the found location
        """

    def ground(self) -> Location:
        """
        Default specialized_designators which returns the first element of the iterator of this instance.

        :return: A resolved location
        """
```