```
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 ActionDesignatorDescription(DesignatorDescription):
    """
    Abstract class for action designator descriptions.
    Descriptions hold possible parameter ranges for action designators.
    """

    @dataclass
    class Action:
        """
        The performable designator with a single element for each list of possible parameter.
        """
        robot_position: Pose = field(init=False)
        """
        The position of the robot at the start of the action.
        """
        robot_torso_height: float = field(init=False)
        """
        The torso height of the robot at the start of the action.
        """

        robot_type: ObjectType = field(init=False)
        """
        The type of the robot at the start of the action.
        """

        def perform(self) -> Any:
            """
            Executes the action with the single parameters from the description.
            """
            raise NotImplementedError()

        def ground(self) -> Action:
            """Fill all missing parameters and chose plan to execute. """
            raise NotImplementedError(f"{type(self)}.ground() is not implemented.")


class MoveTorsoAction(ActionDesignatorDescription):
    """
    Action Designator for Moving the torso of the robot up and down
    """

    def __init__(self, positions: List[float], resolver=None):
        """
            Create a designator description to move the torso of the robot up and down.

            :param positions: List of possible positions of the robots torso, possible position is a float of
height in metres
            """

    def ground(self) -> MoveTorsoActionPerformable:
        """
        Returns a performable designator with the first entries from the list of possible torso heights.

        :return: A performable action designator
        """



class PickUpAction(ActionDesignatorDescription):
    """
    Designator to let the robot pick up an object.
    """

    def __init__(self,
                 object_designator_description: Union[ObjectDesignatorDescription,
ObjectDesignatorDescription.Object],
                 arms: List[str], grasps: List[str], resolver=None):
        """
        Lets the robot pick up an object. The description needs an object designator describing the object
that should be
        picked up, an arm that should be used as well as the grasp from which side the object should be
picked up.

        :param object_designator_description: List of possible object designator
        :param arms: List of possible arms that could be used
        :param grasps: List of possible grasps for the object
        :param resolver: An optional specialized_designators that returns a performable designator with
elements from the lists of possible paramter
```

```
        :param ontology_concept_holders: A list of ontology concepts that the action is categorized as or
associated with
        """

    def ground(self) -> PickUpActionPerformable:
        """
        Returns a performable designator with the first entries from the lists of possible parameter.

        :return: A performable designator
        """


class PlaceAction(ActionDesignatorDescription):
    """
    Designator to place an Object at a position using an arm.
    """

    def __init__(self,
                 object_designator_description: Union[ObjectDesignatorDescription,
ObjectDesignatorDescription.Object],
                 target_locations: List[Pose],
                 arms: List[str], resolver=None, ontology_concept_holders: Optional[List[Thing]] = None):
        """
        Create an Action Description to place an object

        :param object_designator_description: Description of object to place.
        :param target_locations: List of possible positions/orientations to place the object
        :param arms: List of possible arms to use
        """

    def ground(self) -> PlaceActionPerformable:
        """
        Returns a performable designator with the first entries from the list of possible entries.

        :return: A performable designator
        """


class NavigateAction(ActionDesignatorDescription):
    """
    Designator to navigates the Robot to a position.
    """

    def __init__(self, target_locations: List[Pose], resolver=None:
        """
        Navigates the robot to a location.

        :param target_locations: A list of possible target locations for the navigation.
        """
    def ground(self) -> NavigateActionPerformable:
        """
        Returns a performable designator with the first entry of possible target locations.

        :return: A performable designator
        """



@dataclass
class ActionAbstract(ActionDesignatorDescription.Action, abc.ABC):
    """Base class for performable performables."""

    @abc.abstractmethod
    def perform(self) -> None:
        """
        Perform the action.

        Will be overwritten by each action.
        """

@dataclass
class MoveTorsoActionPerformable(ActionAbstract):
    """
    Move the torso of the robot up and down.
    """
    position: float
    """
    Target position of the torso joint
    """

    def perform(self) -> None:
        """ Performs the action """


@dataclass
class PickUpActionPerformable(ActionAbstract):
    """
    Let the robot pick up an object.
    """

    object_designator: ObjectDesignatorDescription.Object
    """
    Object designator describing the object that should be picked up
    """

    arm: str
    """
    The arm that should be used for pick up
    """

    grasp: str
    """
    The grasp that should be used. For example, 'left' or 'right'
    """

    object_at_execution: Optional[ObjectDesignatorDescription.Object] = field(init=False)
    """
    The object at the time this Action got created. It is used to be a static, information holding entity. It is
    not updated when the BulletWorld object is changed.
    """

    def perform(self) -> None:
        """ Performs the action"""


@dataclass
class PlaceActionPerformable(ActionAbstract):
    """
    Places an Object at a position using an arm.
    """

    object_designator: ObjectDesignatorDescription.Object
    """
    Object designator describing the object that should be place
    """
    arm: str
    """
    Arm that is currently holding the object
    """
    target_location: Pose
    """
    Pose in the world at which the object should be placed
    """
    def perform(self) -> None:
        """ Performs the action """

@dataclass
class NavigateActionPerformable(ActionAbstract):
    """
    Navigates the Robot to a position.
    """

    target_location: Pose
    """
    Location to which the robot should be navigated
    """
```

```
def perform(self) -> None:
    """ Performs the action """
```