```
from dataclasses import dataclass, field
from typing import Dict, List, Optional, Union

from swarms.structs.agent import Agent
from swarms.structs.base_swarm import BaseSwarm
from swarms.utils.loguru_logger import initialize_logger


logger = initialize_logger("company-swarm")


@dataclass
class Company(BaseSwarm):
    """
    Represents a company with a hierarchical organizational structure.
    """

    org_chart: List[List[Agent]]
    shared_instructions: str = None
    ceo: Optional[Agent] = None
    agents: List[Agent] = field(default_factory=list)
    agent_interactions: Dict[str, List[str]] = field(
        default_factory=dict
    )

    def __post_init__(self):
```

```
    self._parse_org_chart(self.org_chart)


def add(self, agent: Agent) -> None:
    """
    Adds an agent to the company.

    Args:
        agent (Agent): The agent to be added.

    Raises:
        ValueError: If an agent with the same ID already exists in the company.
    """
    try:
        if any(
            existing_agent.id == agent.id
            for existing_agent in self.agents
        ):
            raise ValueError(
                f"Agent with id {agent.id} already exists in the"
                " company."
            )
        self.agents.append(agent)

    except Exception as error:
        logger.error(
            f"[ERROR][CLASS: Company][METHOD: add] {error}"
```

```
        )
        raise error


def get(self, agent_name: str) -> Agent:
    """
    Retrieves an agent from the company by name.

    Args:
        agent_name (str): The name of the agent to retrieve.

    Returns:
        Agent: The retrieved agent.

    Raises:
        ValueError: If an agent with the specified name does not exist in the company.
    """
    try:
        for agent in self.agents:
            if agent.name == agent_name:
                return agent
        raise ValueError(
            f"Agent with name {agent_name} does not exist in the"
            " company."
        )
    except Exception as error:
        logger.error(
```

```
            f"[ERROR][CLASS: Company][METHOD: get] {error}"
        )
        raise error

def remove(self, agent: Agent) -> None:
    """
    Removes an agent from the company.

    Args:
        agent (Agent): The agent to be removed.
    """
    try:
        self.agents.remove(agent)
    except Exception as error:
        logger.error(
            f"[ERROR][CLASS: Company][METHOD: remove] {error}"
        )
        raise error

def _parse_org_chart(
    self, org_chart: Union[List[Agent], List[List[Agent]]]
) -> None:
    """
    Parses the organization chart and adds agents to the company.

    Args:
```

```
    org_chart (Union[List[Agent], List[List[Agent]]]): The organization chart
        representing the hierarchy of agents.

Raises:
    ValueError: If more than one CEO is found in the org chart or if an invalid
        agent is encountered.
"""
try:
    for node in org_chart:
        if isinstance(node, Agent):
            if self.ceo:
                raise ValueError("1 CEO is only allowed")
            self.ceo = node
            self.add(node)

        elif isinstance(node, list):
            for agent in node:
                if not isinstance(agent, Agent):
                    raise ValueError(
                        "Invalid agent in org chart"
                    )
                self.add(agent)

            for i, agent in enumerate(node):
                if i == len(node) - 1:
                    continue
```

```
                for other_agent in node[i + 1]:
                    self.__init_task(agent, other_agent)
    except Exception as error:
        logger.error(
            "[ERROR][CLASS: Company][METHOD: _parse_org_chart]"
            f" {error}"
        )
        raise error


def _init_interaction(
    self,
    agent1: Agent,
    agent2: Agent,
) -> None:
    """
    Initializes the interaction between two agents.

    Args:
        agent1 (Agent): The first agent involved in the interaction.
        agent2 (Agent): The second agent involved in the interaction.

    Returns:
        None
    """
    if agent1.ai_name not in self.agents_interactions:
```

```
        self.agents_interactions[agent1.ai_name] = []
    self.agents_interactions[agent1.ai_name].append(
        agent2.ai_name
    )


def run(self):
    """
    Run the company
    """
    for (
        agent_name,
        interaction_agents,
    ) in self.agents_interactions.items():
        agent = self.get(agent_name)
        for interaction_agent in interaction_agents:
            task_description = (
                f"Task for {agent_name} to interact with"
                f" {interaction_agent}"
            )
            print(f"{task_description} is being executed")
            agent.run(task_description)
```