```
import json
import re
from typing import Type, TypeVar

from pydantic import BaseModel, ValidationError

T = TypeVar("T", bound=BaseModel)


class JsonParsingException(Exception):
    """Custom exception for errors in JSON parsing."""


class JsonOutputParser:
    """Parse JSON output using a Pydantic model.

    This parser is designed to extract JSON formatted data from a given string
    and parse it using a specified Pydantic model for validation.

    Attributes:
        pydantic_object: A Pydantic model class for parsing and validation.
        pattern: A regex pattern to match JSON code blocks.

    Examples:
    >>> from pydantic import BaseModel
    >>> from swarms.utils.json_output_parser import JsonOutputParser
>>> class MyModel(BaseModel):
...     name: str
...     age: int
...
>>> parser = JsonOutputParser(MyModel)
>>> text = "```json\n{\"name\": \"John\", \"age\": 42}\n```"
>>> model = parser.parse(text)
>>> model.name

"""

def __init__(self, pydantic_object: Type[T]):
    self.pydantic_object = pydantic_object
    self.pattern = re.compile(
        r"^```(?:json)?(?P<json>[^`]*)", re.MULTILINE | re.DOTALL
    )

def parse(self, text: str) -> T:
    """Parse the provided text to extract and validate JSON data.

    Args:
        text: A string containing potential JSON data.

    Returns:
        An instance of the specified Pydantic model with parsed data.
```

```
        Raises:
            JsonParsingException: If parsing or validation fails.
        """
        try:
            match = re.search(self.pattern, text.strip())
            json_str = match.group("json") if match else text

            json_object = json.loads(json_str)
            return self.pydantic_object.parse_obj(json_object)

        except (json.JSONDecodeError, ValidationError) as e:
            name = self.pydantic_object.__name__
            msg = (
                f"Failed to parse {name} from text '{text}'."
                f" Error: {e}"
            )
            raise JsonParsingException(msg) from e

    def get_format_instructions(self) -> str:
        """Generate formatting instructions based on the Pydantic model schema.

        Returns:
            A string containing formatting instructions.
        """
        schema = self.pydantic_object.schema()
        reduced_schema = {
            k: v
            for k, v in schema.items()
            if k not in ["title", "type"]
        }
        schema_str = json.dumps(reduced_schema, indent=4)

        format_instructions = (
            f"JSON Formatting Instructions:\n{schema_str}"
        )
        return format_instructions


# # Example usage
# class ExampleModel(BaseModel):
#     field1: int
#     field2: str

# parser = JsonOutputParser(ExampleModel)
# # Use parser.parse(text) to parse JSON data
```