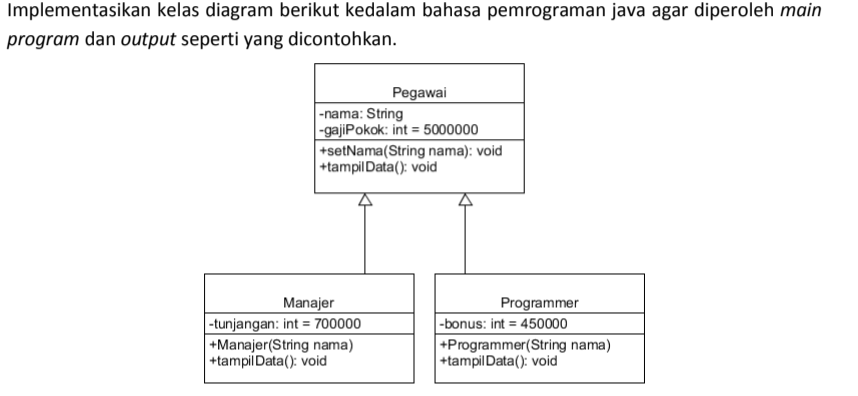
**POST TEST**

**Praktikum Polimorfisme dan Binding**

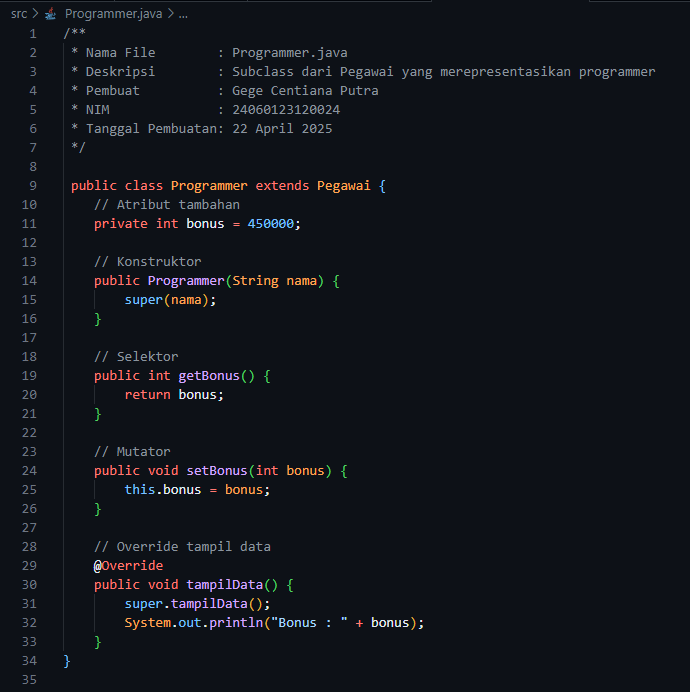
1. Screenshot code dan output dari Study Case.



Source Code pegawai.java sbb:



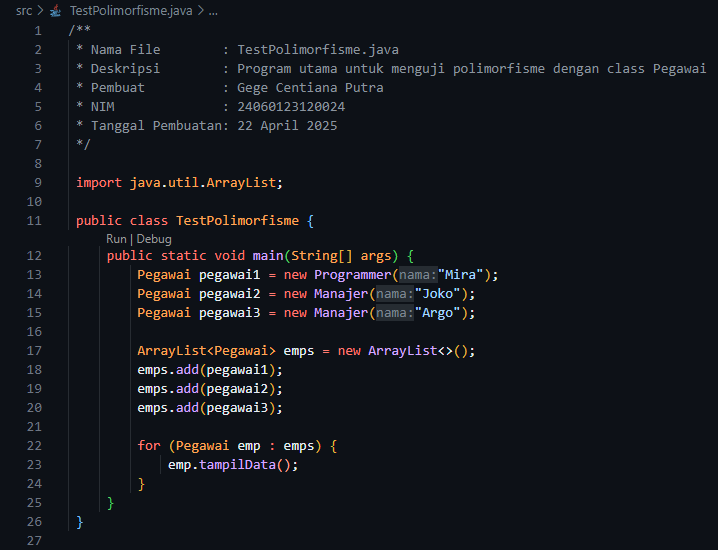
Source code Programmer.java sbb:



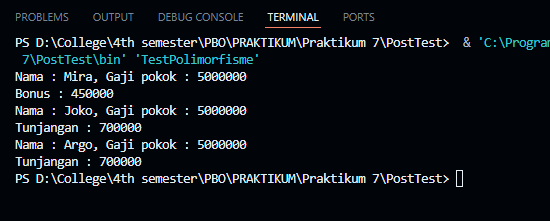
Source code Manajer.java sbb:



Source code Main :



Output Terminal



1. Manfaat Polimorfisme di kasus ini ada beberapa, yaitu:
   1. Koleksi satu tipe tapi bisa banyak bentuk

Kita bisa simpan objek Programmer, Manajer, atau sub­class lain di satu ArrayList<Pegawai> tanpa ribet. Hanya perlu loop, panggil emp.tampilData()—karena override, yang dipanggil method versi masing‑masing.

* 1. Kode ringkas & mudah dikelola

Tanpa harus nulis tampilProgrammer(), tampilManajer(), atau branching if (instanceof), kita cuma panggil satu method. Kalau nanti nambah subtype baru (misal Staff), cukup bikin kelas baru dan override tampilData()—main() tidak perlu diubah.

* 1. Fleksibel jika ada perubahan dan ekstensi

Jika butuh logika khusus di tampilData() , cukup override di subclass, dan tidak perlu sentuh kelas induk atau main(). Jadi skala aplikasi bisa berkembang tanpa “kebablasan” merubah banyak kode.

1. Beberapa permasalahan jika dikelola tanpa polimorfisme, yaitu:

Misalnya di main() kita menambahkan pegawai4 dan pegawai5—misal satu lagi Programmer, satu lagi Manajer, maka:

* 1. Harus membuat tipe koleksi berbeda

Tanpa polimorfisme, mungkin harus punya ArrayList<Programmer> dan ArrayList<Manajer>, terus loop-nya terpisah. Codenya redudan.

* 1. Harus cek tipe manual

Kalau dipaksakan satu list Object atau Pegawai tanpa override, di loop harus cek if (emp instanceof Programmer) … else if (emp instanceof Manajer)… supaya tahu mau cetak apa.

* 1. Sulit dijaga dan rentan *bug*

Setiap kali nambah jenis pegawai baru, harus ubah main()—nulis lagi instanceof, cast, dan panggil method yang tepat. Kalau lupa satu cabang, data jadi tidak tampil atau program error.