Posttest Pemrograman Berorientasi Objek Praktik

Pertemuan ke-2

|  |  |  |
| --- | --- | --- |
| Nama | : | Muhammad Zaki As Shidiqi |
| NPM | : | 5240411230 |
| Kelas | : | VI |

Petunjuk Pengerjaan : Jawablah secara singkat dan jelas. Waktu pengerjaan maksimal 25 menit.

Soal No. 1

Buatlah sebuah class Mahasiswa dengan atribut nama dan nim, serta method tampilkan\_info(). Buat 1 objek dan panggil method tersebut.

|  |
| --- |
| Jawab: |
| class Mahasiswa:      def \_\_init\_\_(self, nama, nim):          self.nama = nama          self.nim = nim      def tampilkan\_info(self):          print(f"Nama: {self.nama}, NIM: {self.nim}")  mhs1 = Mahasiswa("Zaki", "5240411230")  mhs1.tampilkan\_info() |

Soal No. 2

Sebutkan dan jelaskan secara singkat empat pilar OOP.

|  |
| --- |
| Jawab: |
| 1. Enscapsulation: menyembunyikan detail internal dan hanya mengekspos yang penting saja, seperti panel TV, kita tidak perlu mengetahui isi dari panel tersebut. 2. Inheritance: mewariskan sifat dari class induk(parent class) ke class anak(child class), seperti ikan mewariskan sifat bernafas dengan insang ke anak-anaknya. 3. Polymorphism: perbedaan perilaku dari setiap nama method seperti, method bergerak() akan menjalankan logika untuk **berjalan**. 4. Abstraction: berfokus pada inti/konsep, bukan detail teknis seperti mesin atm, kita hanya perlu tahu cara menggunakannya. |

Soal No. 3

Jelaskan dengan contoh singkat bagaimana konsep Encapsulation bekerja dalam OOP

|  |
| --- |
| Jawab: |
| Encapsulation adalah pilar OOP yang menggabungkan data (atribut) dengan method (fungsi) yang mengoperasikannya ke dalam satu unit (objek), sambil menyembunyikan detail internal yang kompleks dari dunia luar.  class Dompet:      def \_\_init\_\_(self, saldo\_awal):          self.\_\_saldo = saldo\_awal      def tambah\_uang(self, jumlah):          if jumlah > 0:              self.\_\_saldo += jumlah              print(f"Berhasil menambah Rp{jumlah}")          else:              print("Jumlah tidak valid.")      def cek\_saldo(self):          print(f"Saldo Anda saat ini: Rp{self.\_\_saldo}")  dompet\_saya = Dompet(50000)  dompet\_saya.cek\_saldo()  dompet\_saya.tambah\_uang(20000)  dompet\_saya.cek\_saldo() |

Soal No. 4

Buatkan contoh perbandingan kode prosedural vs OOP untuk kasus sederhana: menampilkan data sebuah buku (judul dan penulis).

|  |
| --- |
| Jawab: |
| Prosedural: buku\_judul = "Laskar Pelangi"  buku\_penulis = "Andrea Hirata"  def tampilkan\_buku(judul, penulis):      print(f"Judul: {judul}")      print(f"Penulis: {penulis}")  print("--- Info Buku (Prosedural) ---")  tampilkan\_buku(buku\_judul, buku\_penulis)  OOP:  class Buku:      def \_\_init\_\_(self, judul, penulis):          self.judul = judul          self.penulis = penulis      def tampilkan\_info(self):          print(f"Judul: {self.judul}")          print(f"Penulis: {self.penulis}")  print("--- Info Buku (OOP) ---")  buku1 = Buku("Laskar Pelangi", "Andrea Hirata")  buku1.tampilkan\_info() |

Soal No. 5

Mengapa OOP dianggap lebih baik dalam pengembangan program berskala besar dibandingkan pemrograman prosedural? Jelaskan dengan alasan logis.

|  |
| --- |
| Jawab: |
| 1. Kode Lebih Rapi (Modularitas): objek membungkus data dan fungsinya sendiri, sehingga tidak tercampur dan mudah dikelola. 2. Hemat Waktu (Reusability): class yang sudah ada bisa diwariskan untuk membuat class baru tanpa menulis ulang semua kode. 3. Mudah Diperbaiki (Maintainability): jika ada error, perbaikan cukup dilakukan pada objek yang bermasalah tanpa mengganggu bagian lain. 4. Gampang Dikembangkan (Scalability): Fitur baru dapat ditambahkan dengan membuat objek baru, tanpa harus mengubah kode lama yang sudah stabil. |