Lernzettel

Vererbung, Polymorphie und abstrakte Klassen: Konzepte und Anwendungsbeispiele

Universität: Technische Universität Berlin
Kurs/Modul: Einführung in die Informatik
Erstellungsdatum: September 19, 2025

Lernzettel: Vererbung, Polymorphie und abstrakte Klassen: Konzepte und Anwendungsbeispiele

(1) Grundbegriffe der Objektorientierung.
Objektorientierte Programmierung (OOP) arbeitet mit Klassen und Objekten. Kernbegriffe: Abstraktion, Kapselung, Vererbung, Polymorphie. Vererbung ermöglicht Code-Wiederverwendung durch Ableiten einer neuen Klasse von einer bestehenden Klasse. Polymorphie erlaubt, dass verschiedene Klassen über denselben Typ angesprochen werden und zur Laufzeit die korrekte Methodenausführung erfolgt.

(2) Vererbung.
Eine abgeleitete Klasse (Subklasse) erbt Attribute und Verhalten von der Basisklasse (Superklasse). Vorteile: Wiederverwendung, zentrale Stelle der Änderung, bessere Wartbarkeit. Wichtige Begriffe: Basisklasse, abgeleitete Klasse, Sichtbarkeiten (public, protected, private). In Java: `class Subklasse extends Basisklasse`; in C++: `class Subklasse : public Basisklasse`.

(2a) Beispiel (Java-/C++-artige Darstellung).

```
abstract class Tier {
  protected String name;
  public Tier(String n) { name = n; }
  public abstract void bewegen();
}
class Hund extends Tier {
  public Hund(String n) { super(n); }
  public void bewegen() { System.out.println("laufen"); }
}
```

(2b) Beispiel (Alternative kompakter Java-Skizze).

```
abstract class Tier {
  String name;
  abstract void bewegen();
}
class Katze extends Tier {
  void bewegen() { System.out.println("schleichen"); }
}
```

(3) Polymorphie.
Polymorphie bedeutet, dass Objekte unterschiedlicher Klassen über einen gemeinsamen Typ angesprochen werden können und die konkrete Methode zur Laufzeit entschieden wird (dynamische Bindung). Beispiel: Eine Funktion, die einen Tier-Typ akzeptiert, ruft t.bewegen() auf und erhält das Verhalten der konkreten Unterklasse.

(3a) Beispiel – dynamische Bindung.

```
void zeigeBewegung(Tier t) {
  t.bewegen(); // zur Laufzeit wird die Methode der konkreten Klasse ausgeführt
}

Tier h = new Hund("Bello");
Tier k = new Katze("Minka");

zeigeBewegung(h); // läuft
zeigeBewegung(k); // schleichen
```

(4) Abstrakte Klassen.
Eine Klasse, die mindestens eine abstrakte Methode enthält, wird als abstrakt bezeichnet. Abstrakte Klassen können nicht direkt instanziiert werden; sie dienen zur Strukturierung und als Vorlage für konkrete Unterklassen, die alle abstrakten Methoden implementieren müssen.

(4a) Beispiel – abstrakte Klasse.

```
abstract class Form {
  abstract double flaeche();
}
class Kreis extends Form {
  double r;
  Kreis(double r) { this.r = r; }
  double flaeche() { return Math.PI * r * r; }
}
class Quadrat extends Form {
  double a;
  Quadrat(double a) { this.a = a; }
  double flaeche() { return a * a; }
}
```

(4b) Vorteil abstrakter Klassen.
Durch eine generische Sammlung von Shape-Objekten kann code-smart uniform verarbeitet werden:

```
Form s1 = new Kreis(2.0);
Form s2 = new Quadrat(3.0);
double insgesamt = s1.flaeche() + s2.flaeche();
```

(5) Praktische Hinweise zur Umsetzung.
- Verwende Vererbung, um gemeinsame Eigenschaften in einer Basisklasse zu bündeln.
- Nutze abstrakte Klassen, wenn es sinnvoll ist, eine gemeinsame Schnittstelle zu definieren, aber konkrete Implementierungen zu delegieren.
- Setze Polymorphie dort ein, wo Du über verschiedene Subtypen hinweg gleichartige Vorgänge durchführen willst.
- Beachte Sichtbarkeiten: public für Schnittstellen, protected für zugreifbare Basisklassen-Teile, private für Verstecktes.

(6) Entwurfsschema – Schritt-für-Schritt.
1) Identifiziere gemeinsame Merkmale als Basisklasse.
2) Leite spezialisierte Klassen ab und überschreibe abstrakte oder virtuelle Methoden.
3) Entscheide, ob eine abstrakte Klasse sinnvoll ist oder ob Interfaces/Verträge ausreichen.
4)Nutze Polymorphie, um Code flexibel auf neue Unterklassen zu erweitern.
5) Teste mit Objektsammlungen (z. B. List<Form>), die unterschiedliche Unterklassen enthalten.

(7) Übungsfragen (Kurzfragen).
- Was bedeutet Polymorphie in der Praxis?
- Wann setzt man eine abstrakte Klasse statt einer konkreten Basisklasse ein?
- Wie lässt sich in Java/C++ polymorphes Verhalten demonstrieren?
- Welche Probleme können bei exzessiver Vererbungs-Hierarchie auftreten, und wie kann man sie vermeiden?

Hinweis zur Umsetzung in Java/C++ (Zusammenfassung). - Vererbung: `extends` (Java) bzw. `: public` (C++).
- Abstrakte Klasse: Java `abstract class`, C++ `class ... public: virtual ... = 0;` .
- Polymorphie: virtuelle Methoden in der Basisklasse, Überschreibung in Unterklassen, dynamische Bindung.