Mälardalens Högskola
Akademin för Innovation, Design och Teknik

Objektorienterad programmering

Måndag 20 mars, 14:10–18:30
Lärare: Daniel Hedin, 021-107052 (15:00–17:00)
Två blad handskrivna anteckningar (båda sidor får användas)

Tentamen består av 41 poäng fördelat på 12 uppgifter.
Betygsgränser: 3: 21p, 4: 29p, 5: 35p.

- Skriv tydligt; svårläsliga lösningar underkänns. Du kan skriva på engelska eller svenska.
- Om du inte förstår beskrivningen av en uppgift är du skyldig att fråga.
- Var noga med att redogöra för hur du resonerat och vilka antaganden du har gjort. Detta kan bidra till att du får poäng på din lösning även om den inte är helt korrekt.

Encapsulation

1) Förklara begreppet encapsulation och förklara varför det är viktigt inom objektorienterad programmering. (3 poäng)

2) Vilka språkkonstruktioner finns tillgängliga i C# och C++ för att uttrycka encapsulation? Förklara vad de olika konstruktionerna innebär. (3 poäng)

Klasser

3) Skapa en objektorienterad design av en enkellänkad lista. Uppgiften kan lösas med en eller flera klasser. Du behöver inte implementera metoderna; det räcker att beskriva både det publika och det privata interfacet. (4 poäng)

4) I figurerna 1 och 2 definieras klassen Shape (i C# resp. C++) som representerar en geometrisk figur med en position och en hastighet. Tanken är att det enda sättet att flytta på figuren ska vara att anropa metoden Move(). Därför har positionen deklarerats som privat. Samtidigt vill man kunna läsa av positionen på figuren; därför har även en get-metod definierats. Tyvärr uppnår denna lösning inte det man vill. Visa att det går att flytta på figuren utan att använda metoden Move() och föreslå en lösning som inte tillåter detta samtidigt som den tillåter att man läser av positionen. Du kan ge ditt svar som C#-kod eller C++-kod. Förklara varför din lösning fungerar. (3 poäng)

```
class Vector {
  public int x,y;

  public Vector(int x, int y) {
    this.x = x; this.y = y;
  }
}

class Shape {
  private Vector pos;
  private Vector speed;

  public Vector GetPosition() { return pos; }
  public void Move() { pos = pos + speed; }
}
```

Figur 1: Klassen Shape i C#

```
class Vector {
public:
  int x,y;

  Vector(int x, int y) {
    this->x = x; this->y = y;
  }
};

class Shape {
private:
  Vector pos;
  Vector speed;

public:
  Vector &GetPosition() { return pos; }
  void Move() { pos = pos + speed; }
};
```

Figur 2: Klassen Shape i C++

Arv

5) Vad är skillnaden mellan en abstrakt klass och ett interface? När bör den ena eller den andra användas? (4 poäng)

6) I Figur 3 definieras Square (kvadrat) som en underklass till Rectangle. Beskriv för- och nackdelar med denna lösning. (3 poäng)

```
class Rectangle {
  int width, height;

  public void SetSize(int width, int height) {
    this.width = width; this.height = height;
  }
}

class Square : Rectangle {
  public void SetSize(int size) {
    SetSize(size,size);
  }
}
```

Figur 3: Klasserna Rectangle och Square i C#

Dynamisk bindning

7) Förklara skillnaden mellan early (static) och late (dynamic) binding. (3 poäng)

8) Vad skrivs ut av programmet i Figur 4? (4 poäng)

```
class Base {
public:
  void Method1() { cout << "Base::Method1()" << endl; }
  virtual void Method2() { cout << "Base::Method2()" << endl; }
};

class Derived : public Base {
public:
  void Method1() { cout << "Derived::Method1()" << endl; }
  virtual void Method2() { cout << "Derived::Method2()" << endl; }
};

int main() {
  Derived *o1 = new Derived;
  Base *o2 = o1;
  o1->Method1();
  o1->Method2();
  o2->Method1();
  o2->Method2();
  return 0;
}
```

Figur 4: Ett program som använder dynamisk bindning i C++

Överlagring

9) Överlagra additionsoperatorn (+) för klassen Vector i Figur 1 eller Figur 2 så att implementationen av metoden Move() fungerar som det är tänkt. Det räcker med att du gör det för ett av språken. (3 poäng)

10) Hur och när avgörs det vilken metod som ska anropas om det finns fler än en metod med samma namn? (3 poäng)

Generics och templates

11) Skapa en generisk stack med operationerna Push(), Pop() och Peek(). Push() lägger till ett element på toppen av stacken, Pop() plockar bort elementet på toppen av stacken och returnerar det, och Peek() returnerar elementet på toppen av stacken utan att ta bort det. Du kan välja att implementera din lösning i endera C# eller C++. För full poäng skall din implementation generera exceptions om en operation inte kan utföras. (6 poäng)

12) Till skillnad från generics i C# instantieras templates i C++ vid kompilering. Vilka begränsningar innebär detta? (2 poäng)