1. **Cand decidem sa utilizăm membri statici (date si metode) intr-o clasa? Dati un mic exemplu.**

   Membrii statici (date și metode) se utilizează atunci când dorim ca aceștia să fie comuni pentru toate instanțele unei clase, adică să nu fie legați de o instanță specifică a clasei, ci de întreaga clasă.

```cpp
class ContBancar {
private:
    static int numarConturi;  // Membru static

public:
    ContBancar() {
        numarConturi++;  // Crește la fiecare obiect creat
    }

    static int getNumarConturi() {  // Metodă statică
        return numarConturi;
    }
};

int ContBancar::numarConturi = 0;  // Inițializare statică

int main() {
    ContBancar c1, c2, c3;
    cout << "Număr conturi: " << ContBancar::getNumarConturi() << endl;
    // Output: 3
}
```

4. Dați un exemplu de o clasa abstracta. Care este utilitatea ei?

O clasă abstractă este o clasă care conține cel puțin o metodă pur virtuală (`= 0`) și nu poate fi instanțiată direct. Se folosește pentru a defini o interfață comună pentru clasele derivate.

Utilitatea unei clase abstracte:

Abstracție – Definim un concept general (ex. `Forma`) fără a implementa detalii specifice**.**
Polimorfism – Putem lucra cu obiecte de tipuri diferite (`Cerc`, `Patrat`) folosind același pointer (`Forma*`).

Extensibilitate – Putem adăuga ușor noi forme (`Dreptunghi`, `Triunghi` etc.), fără a schimba codul existent.

```cpp
// Clasa abstracta
class Animal {
public:
    // Metoda virtuala pura
    virtual void vorbeste() = 0; // Aceasta este metoda pur virtuala
};

// Clasa derivata
class Caine : public Animal {
public:
    void vorbeste() override {
        cout << "Ham, ham!" << endl;
    }
};

int main() {
    Caine caine;
    caine.vorbeste(); // Afiseaza: Ham, ham!

    return 0;
}
```

6. Utilitatea constructorului de copiere. Un scurt exemplu.

Folosim constructorul de copiere atunci când avem nevoie să creăm un nou obiect care să fie o copie exactă a unui alt obiect deja existent.

```cpp
class Persoana {
private:
    string nume;
    int varsta;
```

```
    Persoana(const Persoana& p) {
        nume = p.nume;
        varsta = p.varsta;
        cout << "Constructor de copiere apelat!" << endl;
    }
```

7. Cum este utilizat sistemul de tratare a excepțiilor in C++. Un scurt exemplu.

Copy Edit

```
class Exceptie {
private:
    char catEx[50];
    char tipEx[50];

public:
    // Constructor
    Exceptie(const char* _catEx, const char* _tipEx) {
        strcpy(catEx, _catEx);
        strcpy(tipEx, _tipEx);
    }

    // Metoda pentru afisarea detaliilor exceptiei
    void Print() const {
        cerr << "Exceptie [" << catEx << "] - " << tipEx << endl;
    }
};

// Functie care arunca o exceptie
void divizare(int numitor) {
    if (numitor == 0) {
        throw Exceptie("Matematica", "Impartire la zero");
    } else {
        cout << "Rezultatul impartirii: " << 10 / numitor << endl;
    }
}

int main() {
    try {
        divizare(0);  // Va arunca exceptia
    } catch (const Exceptie& e) {  // Prinderea si gestionarea exceptiei
        e.Print();  // Afisarea detaliilor exceptiei
    }

    return 0;
}
```