



#include <iostream>

Encapsulation

```

class Heart {
public:
    void pump() {
        std::cout << "Pumping blood" << std::endl;
    }
};

class Human {
private:
    Heart heart; // Encapsulated heart

public:
    void pumpBlood() {
        heart.pump(); // Only accessible through this method
    }
};

int main() {
    Human person;
    person.pumpBlood(); // We interact with the heart through the human
    return 0;
}

```

#include <iostream>

Inheritance

```

class Human {
public:
    void walk() {
        std::cout << "Walking" << std::endl;
    }
};

class Athlete : public Human {
public:
    void run() {
        std::cout << "Running fast" << std::endl;
    }
};

int main() {
    Athlete athlete;
    athlete.walk(); // Inherited method
    athlete.run(); // New method
    return 0;
}

```

```
#include <iostream>
```

Polymorphism :-

```
class Human {
public:
    virtual void move() {
        std::cout << "Moving" << std::endl;
    }
};

class Athlete : public Human {
public:
    void move() override {
        std::cout << "Running fast" << std::endl;
    }
};

class Dancer : public Human {
public:
    void move() override {
        std::cout << "Dancing gracefully" << std::endl;
    }
};

int main() {
    Human* people[] = { new Athlete(), new Dancer() };

    for (Human* person : people) {
        person->move(); // Different behaviors depending on the object
    }

    // Cleanup
    for (Human* person : people) {
        delete person;
    }

    return 0;
}
```

```
#include <iostream>
```

Abstraction

```
class Human {
private:
    void moveLegs() {
        std::cout << "Legs are moving" << std::endl;
    }
public:
    void walk() {
        moveLegs(); // We abstract the complex details
    }
};

int main() {
    Human person;
    person.walk(); // We don't need to know how the legs move
    return 0;
}
```