**Slip 11: Sample Solutions and Explanations**

**Q1. Exception: Throw INVALID AGE**

**Approach**

* Check if the entered age is valid (between 0 and 100).
* If not, throw and handle an exception with a custom error message.

**Code**

#include <iostream>  
#include <stdexcept>  
using namespace std;  
  
// [Age Validation Function]  
void checkAge(int age) {  
 if(age < 0 || age > 100) throw runtime\_error("INVALID AGE");  
}  
  
int main() {  
 int age;  
 cout << "Enter age: ";  
 cin >> age;  
 try {  
 checkAge(age);  
 cout << "Valid age.\n";  
 } catch(runtime\_error &e) {  
 cout << e.what() << endl;  
 }  
 return 0;  
}

**Explanation**

* The checkAge function throws a runtime\_error if the age is not in the valid range.
* The exception is caught in main and the error message is displayed.

**Syntax Definitions**

* **throw**: Used to signal the occurrence of an exception.
* **try-catch**: Used to handle exceptions and prevent program termination.
* **runtime\_error**: A standard exception class for runtime errors.

**Q2. Temperature Converter with Static Functions**

**Approach**

* Create a class with static functions to convert between Celsius and Fahrenheit.
* Call these functions directly using the class name.

**Code**

#include <iostream>  
using namespace std;  
  
// [Temperature Converter Class]  
class TConverter {  
public:  
 static double CtoF(double c) { return c\*9/5 + 32; }  
 static double FtoC(double f) { return (f-32)\*5/9; }  
};  
  
int main() {  
 double c, f;  
 cout << "Celsius: "; cin >> c;  
 cout << "Fahrenheit: " << TConverter::CtoF(c) << endl;  
 cout << "Fahrenheit: "; cin >> f;  
 cout << "Celsius: " << TConverter::FtoC(f) << endl;  
 return 0;  
}

**Explanation**

* The TConverter class provides static methods for temperature conversion.
* Static methods are called using the class name, without creating an object.

**Syntax Definitions**

* **static**: Declares a member function or variable that belongs to the class, not to any object.

**Q3. Online Retail Store Product Management (Case Study)**

**Approach**

* Define a Product class with id, name, price, category, and stock.
* Allow adding, updating, and retrieving products by category or price range.

**Code**

#include <iostream>  
#include <vector>  
using namespace std;  
  
// [Product Class Definition]  
class Product {  
 int id, stock;  
 string name, category;  
 double price;  
public:  
 void accept() {  
 cout << "ID:"; cin >> id;  
 cout << "Name:"; cin >> name;  
 cout << "Category:"; cin >> category;  
 cout << "Price:"; cin >> price;  
 cout << "Stock:"; cin >> stock;  
 }  
 void update(double p, int s) { price = p; stock = s; }  
 void display() { cout << id << " " << name << " " << category << " " << price << " " << stock << endl; }  
 bool matchCategory(string cat) { return category == cat; }  
 bool matchPrice(double lo, double hi) { return price >= lo && price <= hi; }  
};  
  
int main() {  
 vector<Product> prods;  
 for(int i = 0; i < 2; ++i) { Product p; p.accept(); prods.push\_back(p); }  
 cout << "Category search: "; string cat; cin >> cat;  
 for(auto &p : prods) if(p.matchCategory(cat)) p.display();  
 cout << "Update first product...\n"; prods[0].update(99.99, 100); prods[0].display();  
 return 0;  
}

**Explanation**

* The Product class manages product details and provides methods for updating and searching.
* Products are stored in a vector, and can be filtered by category or price range.
* Demonstrates basic CRUD operations for a retail system.

**Syntax Definitions**

* **vector**: A dynamic array from the C++ Standard Library.
* **Method**: A function defined inside a class.