Abstract Classes and Interfaces

Abstract Classes and Interfaces

- Some classes have no instances
- They are superordinate concepts having a given specification
- Basic concepts give ‘body’ to these classes
- They tend to be base classes and may have attributes

2

Why ABCs?

- Placeholders for a hierarchy of more specific classes
- Client code use ABC in functions but ABC ‘refers’ to a derived class
- Client needs no knowledge of derived classes
- Substitutability principle

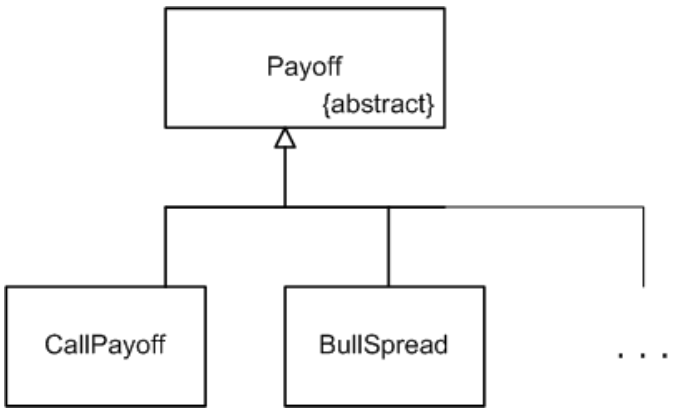
3

Abstract Classes and Interfaces

- Some classes have no instances
- They are superordinate concepts having a given specification
- Basic concepts give ‘body’ to these classes
- They tend to be base classes and may have attributes

4

Example



5

Example: Payoff

```
class Payoff
{ // Superordinate class

public:
  // Pure virtual payoff function
  virtual double payoff(double S) const = 0;
};
```

6

Derived Class

```
class CallPayoff: public Payoff
{
private:
  double K;  // Strike price

public:

  // Constructors and destructor
  CallPayoff();                   // Default constructor
  CallPayoff(double strike);

  // Implement the pure virtual payoff function from base class
  double payoff(double S) const;  // For a given spot price
};
```

7

Interface

- A specification of a set of ‘pure’ functions
- Corresponds to the *superordinate* levels of conceptual hieracrchies
- The functions have no body
- An interface has no data and no non-abstract functions
- C++ does not support interfaces, but can simulate with ‘minimal’ ABCs (containing *pure virtual member functions*)

8