Polymorphism

Eden Burton <eden.burton@senecacollege.ca>
github repository:
(https://github.com/Seneca-OOP244/SCD-Notes)

19/11/2015

Polymorphism

".. of many forms .."

recall that types are...

- associated with objects
- used to check the correctness of expressions

polymorphism

- selects an operation based on object type
- types
    1. *ad-hoc*, “pretend” or fake
    2. *universial*, the real deal

Polymorphism Types

Universal Polymorphism

...same function logic applied to different types...

Universal Polymorphism - Inclusion

- selection of a member function definition from a set based on object type
- based on inheritance hierarchy

```
class Account {
    ...
public:
    void withdraw(double amt);  };

class SavingsAccount : public Account { ... };

int main() {
    SavingsAccount bobSavings(...);
    Account jAcct(...);
    bobSavings.withdraw(100) ;
    jAcct.withdraw(100); }
```

Abstract Base Classes

- class without a complete implementation (interface)
- separates interface from implementation
- specify using pure virtual functions
  - `virtual Type funcID(parameters) = 0;`
- concrete classes implement interfaces

Universal Polymorphism - Inclusion (types)

- polymorphic objects, change type throughout its lifetime
- static type, based on reference type
- dynamic type, based on type used to allocate object

```
int main() {
    SavingsAccount bobSavings(...);
    Account& someAcct = bobSavings;
    ...
    // dynamic type of someAcct ???
    // static type of someAcct ????
}
```

Universal Polymorphism - Inclusion (examples)

```
class Account { ...
    void deposit(double amt){...} ;  };

class SavingsAccount : public Account { ...
    void deposit(double amt{...} );  };

int main() {
   SavingsAccount bobSavings(...);
   Account jAcct(...);
   bobSavings.deposit(100);
   jAcct.deposit(100);
   Account& someAcct = bobSavings;

   // which deposit is called ???
   someAcct.deposit(100); }
```

Universal Polymorphism - Inclusion (virtual functions)

```
class Account { ...
    virtual void deposit(double amt)\{...\} ; };

class SavingsAccount : public Account { ...
    void deposit(double amt{...} );  };

int main() {
   SavingsAccount bobSavings(...);
   Account jAcct(...);
   bobSavings.deposit(100);
   jAcct.deposit(100);
   Account someAcct = bobSavings;

   // virtual causes method resolution based
   // on dynamic type
   someAcct.deposit(100); }
```

Universal Polymorphism - Parametric

- separate interfaces from implementation
- clients use same logic using unrelated types
- implemented using templates
- compiler generates multiple copies of functions

Universal Polymorphism - Parametric

Template Syntax

```
template <typename T>

 // ...  template body follows here


T value; // value is of type T
```

Compiler replaces `T` with client argument within body