03 – Functions and Methods

OBJECT-ORIENTED PROGRAMMING

FUNCTIONS AND METHODS

- how arguments are passed to functions:
  - C++:
    - pass by value
    - pass by pointer
    - pass by reference
  - Java:
    - pass by value (*of object reference*)
- function (method) overloading

FUNCTION DECLARATIONS

- Prototype (signature):

```
void credit(int anAmount)
```

- Implementation (C++):

```
class Account
public:
  void credit(int anAmount) {
   theBalance += anAmount;
  }
};
```

PASSING ARGUMENTS IN C++

- three different ways to pass arguments to functions in C++:
  - pass by value
  - pass by pointer
  - pass by reference
- understanding the distinction between "pass by value" in C++ and "pass by value" in Java

PASSING A PRIMITIVE TYPE ARGUMENT BY VALUE

- The called function makes a local copy of the argument object
- Any changes to the local copy are not visible in the calling function

```
void main() {
    int x = 100;
    g(x);                    // A
    cout << x << endl;       // 100
}
void g(int y) { y++;}        // B
```

PASSING A PRIMITIVE TYPE ARGUMENT BY POINTER

- passing a copy of the memory address to the called function
- A called function can be made to bring about changes that are visible in the calling function

```
void main() {
    int x = 100;
    g(&x);                  // A
    cout << x << endl;      // 101
}
void g(int* y) { (*y)++;}  // B
```

PASSING A PRIMITIVE TYPE ARGUMENT BY REFERENCE

- the reference types in C++ serves as merely another name—an alias

```
void main() {
   int x = 100;
   g(x);               // A
   cout << x << endl;  // 101
}
void g(int& y) { y++;} // B
```

PASSING A CLASS TYPE ARGUMENT BY VALUE

- works the same way as passing a primitive type argument

```
void main() {
    User u("Xenon", 89);                          //(A)
    g(u);                                         //(B)
    cout << u.name << " " << u.age << endl;       // Xenon 89
}
void g(User v) {                                  //(C)
    v.name = "Yukon";
    v.age = 200;
}
```

PASSING A CLASS TYPE ARGUMENT BY POINTER

- any changes made to the object inside the called function can become visible in the calling function

```
void main() {
    User* p = new User("Xeno", 89);            //(A)
    g(p);                                      //(B)
    cout << p->name << " " << p->age << endl;  // Xeno 89
    h(p);                                      //(C)
    cout << p->name << " " << p->age << endl;  // Yuki 200
}
void g(User* q) {                               //(D)
    q = new User("Yuki", 200);                 //(E)
}
void h(User* q) {                              //(F)
    q->name = "Yuki";                          //(G)
    q->age = 200;                              //(H)
}
```

PASSING A CLASS TYPE ARGUMENT BY REFERENCE

- The local variable is essentially an alias for the variable in main.

```
void main() {
    User u("Xenon", 89);                          //(A)
    g(u);                                         //(B)
    cout << u.name << " " << u.age << endl;       // Yukon 200
}
void g(User& v) {                                 //(C)
    v.name = "Yukon";
    v.age = 200;
}
```

PASSING ARGUMENTS IN JAVA

- only one mode for passing arguments to methods—pass by value
  - Passing a Primitive Type Argument by Value
  - Passing a Class Type Argument by Value of Object Reference

PASSING A PRIMITIVE TYPE ARGUMENT BY VALUE

- no difference between how a primitive argument is passed by value in C++ and in Java

```
class Test {
    public static void main(String[] args) {
      int x = 100;                    //(A)
      g(x);                           //(B)
      System.out.println(x);          // outputs 100
    }
    static void g(int y) { y++; }     //(C)
}
```

PASSING A CLASS TYPE ARGUMENT BY VALUE OF OBJECT REFERENCE

- different from passing a class type argument by value in C++
- similar to the case of passing a class type argument by pointer in C++

```
class Test {
    public static void main(String [] args) {
      User u = new User( "Xeno", 89);               //(A)
      g(u);                                          //(B)
      System.out.println(u.name + " " + u.age);    // Yuki 200
    }
    static void g(User v) {                          //(C)
      v.name = "Yuki";                               //(D)
    v .age = 200;                                    //(E)
    }
}
```

PASS-ARGUMENT-BY-VALUE-OF-OBJECT-REFERENCE IN JAVA VS. PASS-ARGUMENT-BY-REFERENCE IN C++

- the argument passing mode in Java does not at all work like the pass-by-reference mode in C++

```
class Test {
    public static void main(String[] args) {
        User u1 = new User("Xeno", 95);          //(A)
        User u2 = new User("Yuki", 98);          //(B)
        swap(u1, u2);                            //(C)
        System.out.println(u1.name);             // Xeno
        System.out.println(u2.name);             // Yuki
    }
    static void swap(User s, User t) {           //(D)
        User temp = s;
        s = t;
        t = temp;
    }
}
```

PASS-ARGUMENT-BY-VALUE-OF-OBJECT-REFERENCE IN JAVA VS. PASS-ARGUMENT-BY-REFERENCE IN C++ (CONT.)

```
void main() {
    User u1("Xeno", 95);                //(A)
    User u2("Yuki", 98);                //(B)
    swap(u1, u2);                       //(C)
    cout << u1.name << endl;            // Yuki
    cout << u2.name << endl;            // Xeno
    return 0;
}
void swap(User& s, User& t) {           //(D)
    User temp = s; s = t; t = temp;
}
```

- The references s and t in line (D) become aliases for the objects u1 and u2 of main

SUMMARY: DIFFERENCES BETWEEN C++ AND JAVA

- *To pass an argument by value in C++*: the parameter of the called function is handed a copy of the argument object in the calling function
- *To pass an argument by value in Java*: the parameter of the called function is handed a copy of the object reference held by the argument
- *To pass an argument by reference in C++ — not possible in Java*: the reference parameter in the called function simply serves as an alias for the argument object in the calling function

FUNCTION OVERLOADING IN C++

- use the same name with a different number and/or types of arguments

```
class Account {
    string theNumber;
    int theBalance;
public:
    Account() {
          theNumber = “ACB123”;
          theBalance = 0;
    }
    Account(string number, int balance) {
          theNumber = number;
          theBalance = balance;
    }
}
```

FUNCTION OVERLOADING IN JAVA

```
class Account {
    private String theNumber;
    private int theBalance;
    public Account() {
          theNumber = “ACB123”;
          theBalance = 0;
    }
    public Account(String number, int balance) {
          theNumber = number;
          theBalance = balance;
    }
}
```

OBJECT DESTRUCTION

- When objects go out of scope in C++, they are automatically destroyed by the invocation of their destructors
- A destructor is given the name of the class prefixed with a ~
- If no variables in a Java program are holding references to an object, that object becomes a candidate for what is known as *garbage collection*