SSE3052: Embedded Systems Practice

Jinkyu Jeong
jinkyu@skku.edu
Computer Systems Laboratory
Sungkyunkwan University
http://csl.skku.edu

Agenda

- Learning Java!
  - Interface
  - Exception handling
  - ArrayList

Interface

- Declares a set of methods and their signatures
- Unlike class
  - Provides no implementation
  - Cannot instantiate
  - Does not contain any constructor

Ex)

```
public interface Worker {
    double computePay(int hours);
}
```

implement for *interface*

```
public class HourlyWorker implements Worker {
    private double rate;

    public HourlyWorker(double rate) {
        this.rate = rate;
    }

    public double computePay(int hours) {
        if (hours > 40)
            return (hours - 40) * rate * 1.5 + 40 * rate;
        else
            return hours * rate;
    }
}
```

implement for *interface* cont.

```
public class SalariedWorker implements Worker {
    private int rate;

    public SalariedWorker(int rate) {
        this.rate = rate;
    }

    public double computePay(int hours) {
        if (hours > 40)
            return 40 * rate;
        else
            return hours * rate;
    }
}
```

implement for *interface* cont.

```
public class Main {
    public static void main(String[] args) {
        ~~Worker worker1 = new Worker();~~    // Error
        Worker worker2 = new HourlyWorker(7.5);
        Worker worker3 = new SalariedWorker(10);

        System.out.println("worker2: " + worker2.computePay(40));
        System.out.println("worker3: " + worker3.computePay(80));
    }
}
```

implement for *interface* cont.

```
public class SomeClass {
    public void howMuch(Worker worker) {
        System.out.println("You earn " + worker.computePay(40));
    }
}

// “You earn 300” will printed for worker2
// “You earn 400” will printed for worker3
```

Implementing multiple interfaces

- A class can implement multiple interfaces
  - *Cannot inherit multiple superclasses*

Ex)

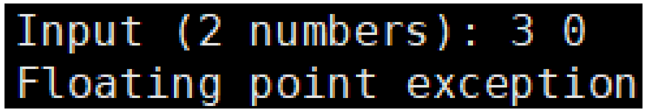
```
public interface AAA {
      public int a();
}
public interface BBB {
      public int b();
}
public class CCC implements AAA,BBB {
      public int a() {…};
      public int b() {…};
}
```

What is Exception?

- An exception is a problem that arises at "runtime"
- e.g., divide by zero

```
int main(void) {
    int num1, num2;
    cout << "Input (2 numbers): ";
    cin >> num1 >> num2;

    cout << "Quotient: " << num1/num2 << endl;
    cout << "Remainder: " << num1 % num2 << endl;

    return 0;
}
```

What if num2 is zero?



```
Input (2 numbers): 3 0
Floating point exception
```

Exception Handling by "if"

- Divide by zero can solved by conditional check

```
int main(void) {
    int num1, num2;
    cout << "Input (2 numbers): ";
    cin >> num1 >> num2;

    if(num2 == 0) {
      cout << "num2 is zero" << endl;
    }
    else {
      cout << "Quotient:" << num1/num2 << endl;
      cout << "Remainder:" << num1 % num2 << endl;
    }
    return 0;
}
```

Exception handling

Then… what is the problem?

Readability

- Developer cannot recognize difference of
  - Logical functionality, and
  - Exception

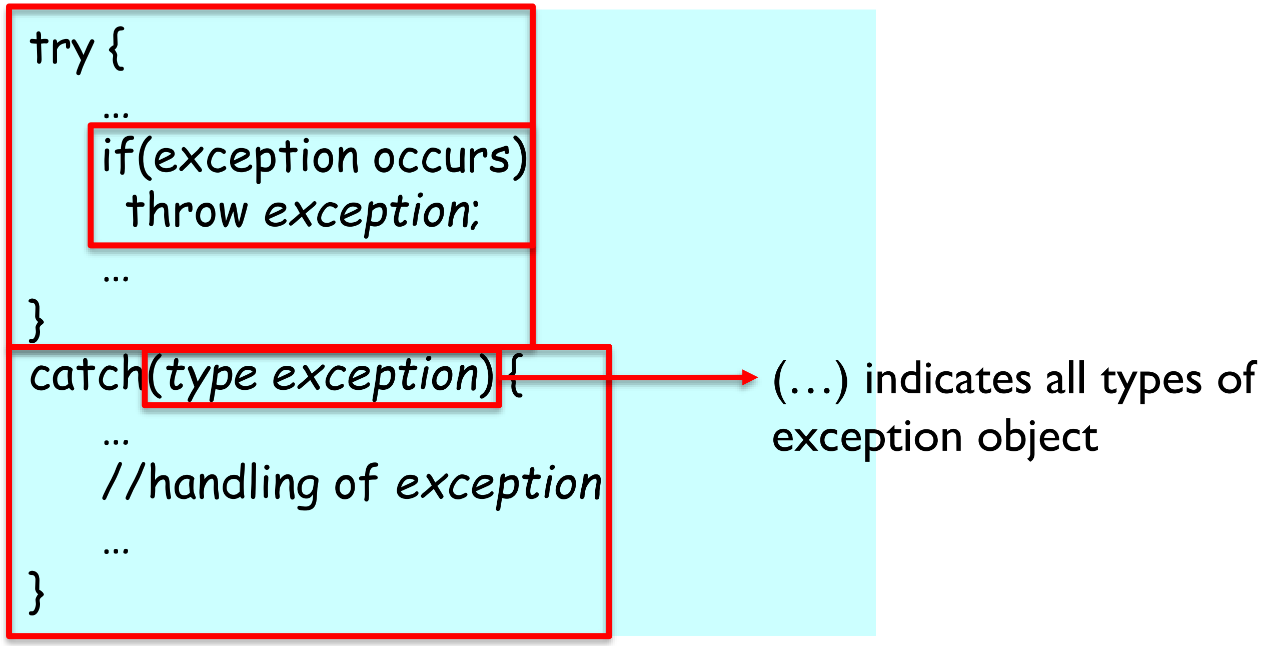
```
int main(void) {
        int num1, num2;
        cout << “Input (2 numbers): “;
        cin >> num1 >> num2;

        if(num2 == 0) {
          cout << “num2 is zero” << endl;
        }
        else if(num2 > num1) {
          cout << “Quotient: 0” << endl;
          cout << “Remainder: “ << num1 << endl;
        }
        else {
          cout << “Division: “ << num1/num2 << endl;
          cout << “Remainder: “ << num1 % num2 << endl;
        }
        return 0;
}
```

Exception Handling

Logical Functionality

Exception Handling

- try-throw-catch
  - try: detect an exception
  - throw: raise an exception
  - catch: handling exception

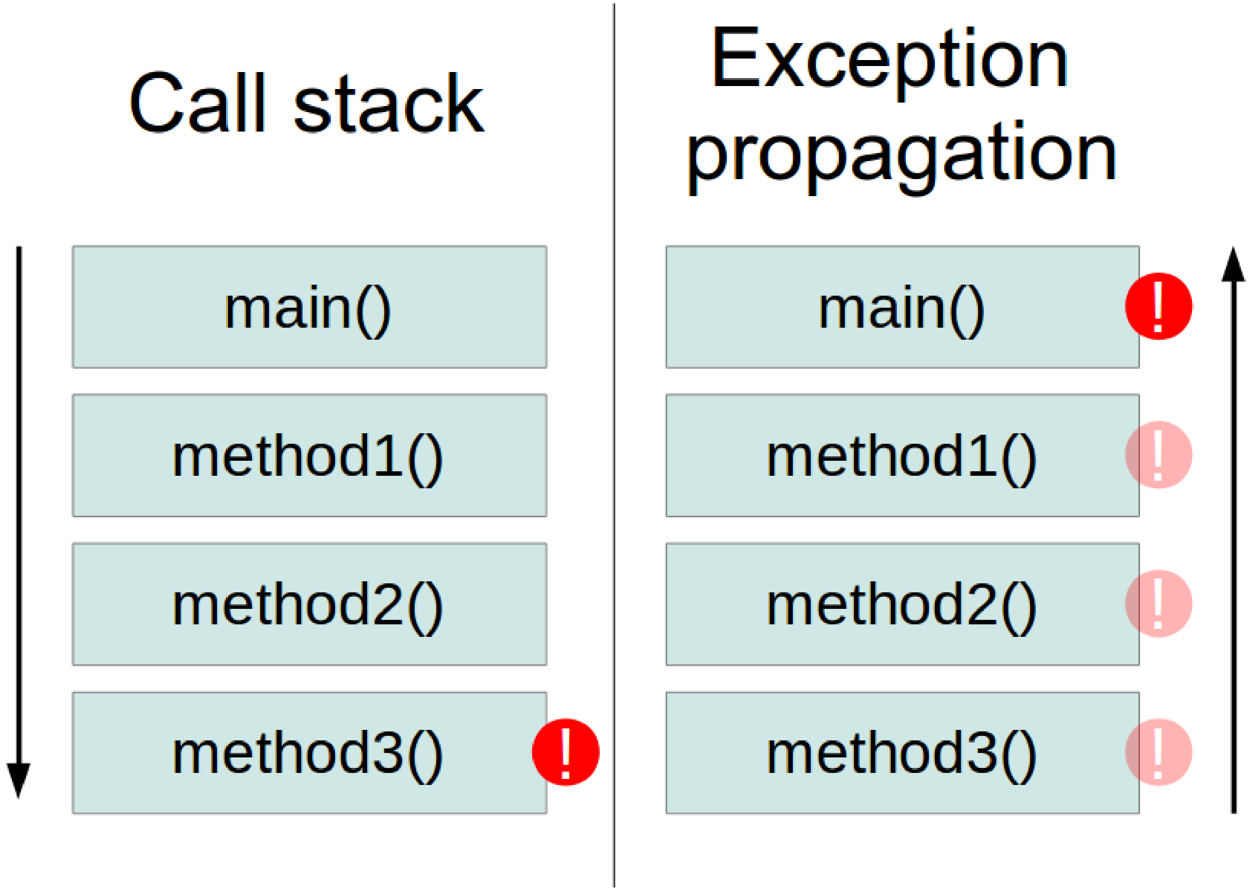


```
try {
      …
      if(exception occurs)
        throw exception;
      …
}
catch(type exception) {
      …
      //handling of exception
      …
}
```

Format of try-throw-catch

Stack Unwinding (cont'd)

- What if there is no *catch* block inside function?
  - Try to find *catch* block of caller



Types of exceptions

- Checked exception
  - Checked at compile-time
  - All subclasses of *Exception* except *RuntimeException*
- Unchecked Exception
  - Not checked at compile-time rather checked at runtime
  - Subclasses of *RuntimeException*

Ex) ArithmeticException

Int a = 50 / 0;

Ex) NullPotinerException

String s = null;

System.out.println(s.length);

Ex) ArrayIndexOutofBoundsException

Int a[] = new int [5];

a[10] = 50;

Throwing exception

```
public class BankAccount {
    public void withdraw(double amount) {
        if (amount > balance) {
            //
            // now what?
        }
        balance = balance - amount;
    }
}
```

Throwing exception cont.

```
public class BankAccount {
    public void withdraw(double amount) {
        if (amount > balance) {
            IllegalArgumentException exception = new IllegalArgum
entException("Amount exceeds balance");
            throw exception;
        }
        balance = balance - amount;
    }
}
```

Catching exception

```
public static void main(String[] args) {
    BankAccount acct = new BankAccount(100);
    try {
        acct.withdraw(200);
    } catch (IllegalArgumentException ex) {
        System.out.println("Withdraw failed.");
        //or ex.printStackTrace();
    }
}
```

Exercise 1

- Modify the *BankAccount* class to throw *IllegalArgumentException* when the account is constructed with a negative balance, when a negative amount is deposited, or when an amount that is not between 0 and the current balance is withdrawn. Write a test program that causes all three exceptions to occur and that catches them all

ArrayList

- An *ArrayList* is a sequence of objects

Ex) Assume that *Coin* class is already defined

```
ArrayList coins = new ArrayList();
coins.add(new Coin(0.1,“dime”));
coins.add(new Coin(0.25,“quarter”));
```

ArrayList methods

- boolean add(E e)
  - Appends the specified element to the end of this list.
- void add(int index, E element)
  - Inserts the specified element at the specified position in this list.
- E get(int index)
  - Returns the element at the specified position in this list.
- E remove(int index)
  - Removes the element at the specified position in this list.
- int size()
  - Returns the number of elements in this list.

Exercise 2

- Implement a class *Bank* that contains an array list of *BankAccount* objects. Support methods,
  - public void addAccount(double initialBalance)
  - public void deposit(int account, double amount)
  - public void withdraw(int account, double amount)
  - public double getBalance(int account)
- An account number is simply an index into the array list