기초Java언어 기말고사 CheatSheet

Chapter08 : 클래스변수와 클래스메소드

- Class Variable
  - Class의 모든 instance가 공유하는 변수
  - static을 통한 선언
  - instance와 다른 영역에 할당
  - JVM 실행과 동시에 memory에 load > instance는 new 선언시에 load
  - Class variable은 initiator 통한 초기화 불가능 > instance 생성시마다 값 reset
  - Class 외부에서는 class 혹은 instance의 이름 통해 접근
- Class Method
  - instance 생성 없이 사용 가능한 method
  - JVM 실행시 loading
  - Class method 내부에서 instance variable의 control : 비권장 > memory에 load되지 않은 variable에 대한 control 시도
- 사용중인 Class Method
  - `java.lang.*` : JVM 실행시 자동으로 import
  - `System.out.println(...)` : out은 System class의 참조변수, println(...)은 out의 instance method
  - `public static void main(String[] args)` : JVM 실행시 호출되는 method
  - static 초기화 block : JVM 시작시 실행되는 block (실행시간표시)
  - Java에서는 Class의 이름 반드시 붙여서 호출

Chapter09 : 메소드오버로딩

- Method Overloading
  - method의 이름은 동일하나 argument가 다른 경우 > return type는 overloading의 조건 아님
  - 정의되지 않은 경우에는 compile error
  - initiator overloading하여 instance 유형 구분
  - this 이용한 기타 생성자의 호출
- String Class
  - String은 new 로 생성 or 변수처럼 사용
  - println()의 overloading - String or int
  - `.length()` : 문자열 길이 저장

    - ""로 생성 : 같은 문자열은 같은 object
    - `new`로 생성 : 같은 문자열도 다른 object
    - String class는 immutable, 새로운 문자열이 return값
    - switch문의 String 사용 : 주소 및 저장값 비교
- String Class method
    - String Class는 immutable : 참조 안하면 GC
    - `.concat(String)` : 문자열 연결, 새로운 instance > 문자열 + 연산 사용시 compiler가 처리 >> 앞의 concat()부터 처리
    - `.substring(num1, num2)` : [num1,num2)
    - `.equals(String)` : boolean
    - `.compareTo(String)` : int (-1,0,1) - 사전식
    - `.compareToIgnoreCase(String)` : 대소문자 무시
    - `str1 == str2` : 주솟값 비교
    - `String.valueOf(obj)` : 문자열 외 type를 문자열로 변환 > 비문자열 + 연산 사용시 compiler가 처리
    - 문자열 결합 최적화
        - String method 사용시 과도한 instance 생성
        - StringBuilder Class 이용하여 처리 (Compiler)
        - StringBuilder는 mutable, 1개의 instance로 처리, 참조값의 유지 가능
        - `StringBuilder.append()`
        - `StringBuilder.delete()`
        - `StringBuilder.replace()`
        - `StringBuilder.reverse()`
        - `StrungBuilder.substring()`
        - StringBuffer : thread 안정성, but 느림
- 콘솔 입출력
    - `System.out.printf()` : C랑 동일 > `-` : 좌측정렬, `%g` : 실수형 자동선택
    - `.toString()` : print에 instance 참조 전달시 문자열 자동변환, Object class로부터 상속
    - Scanner의 argument는 File, String, InputStream(Keyborad)

Chapter10 : 배열

- 1D array
    - 동일 type의 data 저장

- `Object[] ref = new Object[5];` > ref는 array의 참조값 저장 > 각 ref[i]는 new 통한 instance 생성 필요 (참조 값만 초기화)
- `.length` : array의 길이 정보 저장 (자동 생성)
- 반복문 통한 array 조작 : `.length`와 실제 저장 갯수 차
- Array 초기화

```
int[] arr = new int[5];
int[] arr = new int[] {1,2,3};
int[] arr = {1,2,3};
```

- Array는 argument로 참조값 전달
- 기본 자료형 : 0 / Instance : null로 초기화
- `java.util.Arrays'
  - `fill(int[] a, int val)`
  - `fill(int[] a, int fromIndex, int toIndex, int val)` : [fromIndex,toIndex)
- `java.lang.System.arraycopy(Object src, int srcPos, Object dest, int DestPos, int length)`

- Enhanced for (for-each)

```
for (int e : ar){ // e에 각 element 할당
    System.out.println(e);
}
```

- Multi-dimensional Array
  - 각 row를 저장하는 참조 변수의 array
  - 주소 저장 방식은 최적화
  - 각 row의 element 갯수는 상관없음

Chapter11 : 상속

- Inheritance의 이해
  - 기존 Class에 method와 variable 추가

    - Series of class에 공통적인 규약 정의 가능
    - `super` : Parent class의 속성 사용
    - argument 없는 initiator의 경우에만 별도 호출 없이 자동으로 상위 생성자 실행 > Parent 생성자 실행 후 Child 생성자가 실행
    - Java에서는 단일 상속만 가능 (interface는 다중상속 가능)

- Class variables, Class methods and interitance
    - Class variables & methods : static 선언, JVM 시작시 실행
    - static의 경우 접근수준 허용시 child class에서 접근 가능

- 상속을 위한 두 Class의 관계
    - Child Class는 Parent Class의 모든 속성을 포함해야 함 (IS-A 관계)

- Method Overriding (<-> Method Overlodading)
    - OOP 4대요소 중 Polymorphim에 해당 (다형성)
    - 상위 Class의 참조변수가 하위 Class의 참조 가능 > 이 경우 참조 통해 상위 Class 정보만 접근 가능 > 형 변환을 통해서 바꾸어주어야 함
    - Method Overriding > Parent Class의 참조변수로 공통된 method 호출 시 Child Class의 method로 override되어 실행 > Parent Class의 method 사용하고싶다면 instance 내부에서 호출해야 함 (`super` keyword)
    - Instance Variable의 Overriding은 일어나지 않음

- `instanceof()` 연산자
    - 참조변수가 Class 및 Child Class의 instance이면 true
    - 사용 예시 : Class에 따라 다른 method의 적용

```
public static void wrapBox(Box box){
  if (box instance of GoldPapaerBox){
      ((GoldPaerBox)box).goldWrap();
  }
  else if (box instance of PaperBox){
      ((PaperBox)box).paperWrap();
  }
  else
      box.simpleWrap();
}
```

Chapter12 : 상속의 목적 그리고 인터페이스와 추상클래스

- inheritance의 목적
  - 유사한 구조의 Class의 공통 부분 접근 > "연관된 일련의 Class에 대해 공통적인 규약을 정의 및 적용할 수 있음." > Method Overriding 통해 각 Class에 알맞은 method 적용할 수 있음.
- Object Class와 final 선언
  - Java의 모든 class는 `java.lang.Object`를 상속
  - `java.lang.Object.toString()` : Object의 method를 override
  - `final` : 더 이상 상속 불가능한 Class or Method
  - `@Override` : 상속 Class간 Method overloading 불가 (Compile Error) > 필요시 Override 표시
- Interface의 기본과 그 의미
  - interface : 추상 method만 포함
  - instance의 생성 불가능, 참조변수의 사용 가능
  - interface를 implements하는 경우 모든 method의 구현 필요
  - 다중 구현 가능 (다중 상속 불가)
  - interface의 목적 : 구현 지시 & interface 정보만 제공
- Interface의 구성과 추상 클래스
  - interface method는 public (자동으로)
  - interface variable은 public static final
  - interface 사이에도 상속 가능 (`extends`)
  - default method : 실체가 존재하는 method > interface간 상속을 대신하여 사용 > `default void printCMYK(String doc){ }` : 실체가 있음
  - interface의 static method 정의 가능, default method 통해 사용 가능
  - Marker interface
    - interface는 method가 없음
    - Class의 분류 위해 사용
  - abstract class

        - interface와 동일 역할
        - 하나 이상의 추상 method 보유
        - 추상 class의 instance 생성 불가

```
public abstract class House{
  public abstract void methodOne();
}
```

Chapter 13 : 예외처리와 제네릭

- Java의 Exception

```
try{}
catch(ArithmetricException e){}
```

  - 여러 종류의 예외 처리 가능 ('|' 사용)
  - Throwable Class : 모든 예외의 최상위 Class
    - `public String getMessage()` : 예외의 원인 문자열
  - `public void printStackTrace()` : 예외가 발생한 위치와 호출된 method 정보
  - 예외 지점에서 처리하지 않으면 호출한 method로 예외가 전달됨 (상위 method에서 예외처리 가능)
  - exception의 종류
    - `java.lang.ArithmeticException`
    - `java.util.InputMismatchException`
    - `java.lang.ArrayIndexOutOfBoundsException`
    - `java.lang.ClassCastException` > Child의 매개변수로 Parent를 저장할 때
    - `java.lang.NullPointerException`
  - Java에서 Sleep 구현
    - `Thread.sleep()'
    - milisecond 단위 입력
    - 반드시 try & catch 안에서 사용 : `InterruptedException`
    - `System.CurrentTimeMillis()` : long 정수

- Generic의 이해
  - 잦은 type cast로 인해 오류 발생 가능성 높음

```
class Box<T> {
  private T ob;
public T get(){ return ob; }
}

// In main()
Box<Apple> aBox = new Box<>();
```

  - method 호출 시 형 변환 과정 불필요함
  - 오류 발생시 compile error

- Generic의 기본 문법
  - 다중 매개변수 기반 generic class 정의

```
class DBox<L,R>{
  private L left;
private R right;
public void set(L o, R r){ left = o; right = r;}
}

// In main()
DBox<String, Integer> box = new DBox<String, Integer>();
```

  - 기본 자료형은 Class가 아니므로 Integer Class 생성
  - `T extends Number` : 기본 자료형 & 상속 Class만

Chapter 14 : 자바 GUI 프로그래밍(1)

- GUI : Graphical User Interface
  - Java Swing이 AWT를 inherit하는 관계
  - Component : JButton, JLabel, JCheckBox, JChoice, JList, JMenu, JTextField, JScrollbar, JTextArea, Jcanvas
  - Containter : JFrame, JDialog, JApplet, JPanel, JScrollPane > 최상위 Container : JFrame, JDialog, JApplet

```
import javax.swing.*;
import java.awt.*;
class SomeClass extends JFrame{
  public SomeClaee(){
    this.setSize(width, height);
this.setTitle(title);

JButton button = new Button("ButtonText");
this.add(button);

this.setVisible(); // 항상 마지막에 setVisible 표시
  }
}
// in main
SomeClass x = new SomeClass();
```

- `JFrame` : content pane & menu bar 설정 가능
  - `add(component)`
  - `setLocation(x,y)`, `setSize(width, height)`
  - `setIconImage(IconImage)`
  - `setTitle(title)`
  - `setResizeable(boolean)`
- `setLayout(-- layout object --)`
  - `BorderLayout()` : North / South / East / West / Center의 벽면
    - `this.add(component, BorderLayout.NORTH)` : BorderLayout의 Option 지정
  - Size 지정해도 Panel을 Fill
  - `FlowLayout()` : 좌에서 우로
    - `FlowLayout(int align, int hgap, int vgap)`
    - `FlowLayout.LEADING` : 좌측정렬
    - `FlowLayout.CENTER` : 중앙정렬 > default, 간격 5 px
    - `FlowLayout.TRAILING` : 우측정렬
  - `GridLayout(row, col)` : grid 형태로 배치 > row는 0으로 설정 하면 자동으로 할당, col은 반드시 필요
  - `null` : 절대 위치 배치
  - `setDefaultCloseOperation(JFame.EXIT_ON_CLOSE)` : window 종료시 option 추가
  - Frame 위에 Panel 설정 가능 > 하나의 Frame에는 하나의 Layout만 설정 가능 > 각 component의 위치, size 지정 직접 지정 필요

Chapter 15 : 자바 GUI 프로그래밍(2)

- Event 처리
  - GUI는 Event 기반한 처리가 기본

1. Event Source : Event를 발생시킬 수 있는 component > JButton, JList, JTextField
2. Event Instant (Object) : Event에 대한 모든 정보를 갖는 instance > ActionEvent, ItemEvent
3. Event Listener : Event에 따라 호출되는 method > Interface의 형태, 추상 method의 구현 필요 (ActionListener)

```
class MyListener implements ActionListener{
 public void actionPerformed(ActionEvent e){
 // event instance 전달
 }
}
public class MyFram extends JFrame{
 ...
 public MyFrame(){
     button = new JButton();
     button.addActionListener(new MyListener());
     // Listener Class 전달 및 JButton에 등록
     ...
 }
}
```

  - 3가지 Event 처리 방법

1. 내부 Class 사용

```
class A{
 private int b;
 public A(){ ... }

 class B{
 // Class 내부에 Class 구현
     public B(){ b = 100; }
     // 상위 Class의 variable과 method 제약 없이 사용 가능 - private
 }
}
```

2. Frame Class 내부에서 구현

```
// JFrame을 상속받은 Class 내부
class MyFrame extends JFrame implements ActionListener{
 private JButton button;
 public MyFrame(){
 button = new JButton("Button");
 button.addActionListener(this); // this로 현재 instance 전달
}
public void actionPerformed(ActionEvent e){
 JButton e = (JButton)e.getSource(); // (Source object 저장)
 if (e == button){
     // TODO : 각 Button에 해당하는 behavior
 }
}
}
```

3. 무명 Class 사용

```
// in MyFrame()
 button.addActionListener(
 new ActionListener(){
     public void actionPerformed(){
         // TODO : Action Behavior
     }
 }
);
```