**3/15 – 11 장 java.lang.\*/ java.util.\* 클래스**

**12 장 멀티스레드/동기화**

**3/16,17 – 자바 x**

**git 특상**

**3/18 – 15장 컬렉션 프레임워크**

**13장 제네릭**

**3/19 - 14장 람다식**

**3/ 22 - 18장 입출력**

**3/23 - 18장 네트워크**

**(사이트 url – 과제1개, 시험)**

**15장 722p**

**컬렉션 프레임워크**

**배열 – 여러가지 데이터 저장 구조**

**int i [] = new int[50];**

**1> 동일 타입 데이터만 저장 구조**

**(형변환 가능하면 동일 타입 간주)**

**2> i = new int[100];**

**===> 기존 50개 배열 주소 없어지고 100개 배열 주소 참조**

**배열 크기 정적**

**java.util.Date/Calendar**

**- 배열 한계점 개선 여러가지 데이터 저장 구조 클래스**

**1> 여러가지 타입 + 동일 타입**

**2> 크기 동적 변경 가능**

**3> 자바 틀 – 순서 / 값만 저장, 키와 값의 한쌍, 중복 데이터 저장 가능?**

**collection framework 클래스들**

**-java.util.List 인터페이스 – 메소드 구현 x**

**- 클래스로 다중상속 메소드 오버라이딩 구현**

**-순서 데이터 관리 = 인덱스 사용 가능**

**- 같은 데이터 중복 저장 허용**

**add();**

**ArrayList, LinkedList, Vector, Stack 클래스들**

**add() {......}**

**java.util.Set 인터페이스 – 메소드 구현 x**

**- 클래스로 다중상속 메소드 오버라이딩 구현**

**- 순서 없이 데이터 관리 = 인덱스 사용 불가능**

**- 같은 데이터 중복 저장 불가**

**점수들 – 동점**

**로또번호 6개 저장**

**add();**

**HashSet, TreeSet 클래스들--> 검색속도 빠르다**

**java.util.Map 인터페이스 – 메소드 구현 x**

**- 클래스로 다중상속 메소드 오버라이딩 구현**

**- (키, 값)의 한쌍**

**- 키는 중복 X, 값 중복 O**

**HashMap, Hashtable, Properties**

|  |  |  |
| --- | --- | --- |
| **List** | **Set** | **Map** |
| **데이터=값**  **순서있다**  **중복 데이터 허용**  **ArrayList** | **데이터=값**  **순서없다=index x**  **중복 데이터 불가**  **HashSet** | **데이터=(키,값)**  **키 중복허용x/**  **값 중복허용o**  **순서없다**  **HashMap** |
| **여러가지 데이터 타입 저장 가능**  **동적 크기 변경**  **참조형데이터만 저장 가능**  **int, double ===> Integer클래스/ Double클래스** | | |

**- ArrayList**

**<생성자>**

**int i[] = new int[50];**

**ArrayList list = new ArrayList();**

**==> 최초에 10개 객체 저장 ArrayList 생성**

**모자라면 자동 10개씩 증가.**

**ArrayList list = new ArrayList(5);**

**==> 최초에 5개 객체 저장 ArrayList 생성**

**모자라면 자동 5개씩 증가.**

**ArrayList list = new ArrayList(5, 3);**

**==> 최초에 5개 객체 저장 ArrayList 생성**

**모자라면 자동 3개씩 증가.**

**<메소드>- List 인터페이스 상속 오버라이딩**

**저장**

**add(Object o) 정의 / add(int index, Object o) 정의**

**list.add(100)--> 0 인덱스**

**int 기본형 = Integer 클래스 객체 참조형**

**list.add(3.14)--> 1**

**double 기본형 = Double 클래스 객체 참조형**

**list.add("aaa")-->2**

**수정 – list.set(1, "java")**

**삭제 – list.remove(1)**

**조회**

**데이터갯수 조회 – int<--list.size()**

**데이터저장여부 조회 – boolean <---list.contains("java")**

**데이터저장위치 조회 – int( ? >=0) <---- list.indexOf("java")**

**특정위치저장데이터 조회 - Object <--- list.get(2)**

**--> LinkedList, Vector**

**HashSet**

**add("a")**

**add("b")**

**add("a") 실행 무시**

**iterator() --> 저장 데이터 모음(저장 순서 불일치)**

**HashSet set = new HashSet();**

**===> 길이 동적 유지 / 여러 타입 데이터 저장 가능**

**HashSet<A> set = new HashSet<A>();**

**===> 길이 동적 유지 /동일 타입 데이터 저장 가능**

**HashMap map = new HashMap ();**

**//List Set Map 동일 사용**

**HashMap<String, Integer> map = new HashMap <String, Integer> ();**

**HashMap<String, Integer> map = new HashMap ();**

**map.put("id", 100);**

**map.put("score", 100);**

**map.put("score", 200);==> score key 값을 수정**

**map.get("id");**

**map.size()**

**key (String) value(String [] )**

|  |  |
| --- | --- |
| **초등친구** | **폰, 집, 사무실, 이메일** |
| **고등친구** | **폰, 집, 사무실,** |
| **회사동기** | **폰, 집,** |
| **회사상사** | **폰, 이메일** |

**13장 제네릭-generics**

**1> 미리 사용 데이터 타입 알려주고**

**2> 컴파일러 타입 체크**

**3> 잘못된 형변환 실행시 줄어준다**

**Arrayist<A>**

**HashMap<B, D>**

**HahSet<C>**

**-라이브러리 <T> <K>**

**class Fruit<A>{**

**A a1;**

**}**

**class Apple<A,B> extends Fruit<A>**

**B b1;**