1. Java 언어의 장점으로 볼 수 없는 것은?

* 장점
 - WORA(Write Once Run Anywhere) : 한번 코딩해서 여러곳에 사용.
   (한번 작성된 자바코드는 OS에 상관없이 실행 가능함)
 - 문법이 간결하고 배우기 쉽다.
 - 확장성, 모듈화 , 보안성, 안정성이 매우 뛰어난 언어

* 단점
 - 속도가 느림(하드웨어나 임베디드에 사용하기는 어려운 언어)
 - 하드웨어에 대한 직접적인 커트롤 하기 어려움

2. Java 기본타입 변수 값의 범위가 틀린 것은?

byte (정수, 1바이트) : -2^7 ~ 2^7-1
short (정수, 2바이트) : -2^15 ~ 2^15-1
int (정수, 4바이트) : -2^31 ~ 2^31-1
long (정수, 8바이트) : -2^63 ~ 2^63-1
float (실수, 4바이트) : ???
double (실수, 8바이트) : ???
boolean(블리언, 1바이트) : true, false
char(문자, 2바이트 유니코드) : 'a'

3. Java 기본타입과 기본값이 올바르게 짝지어 지지 않은 것은?

byte (정수, 1바이트) : -2^7 ~ 2^7-1
short (정수, 2바이트) : -2^15 ~ 2^15-1
int (정수, 4바이트) : -2^31 ~ 2^31-1
long (정수, 8바이트) : -2^63 ~ 2^63-1
float (실수, 4바이트) : ???
double (실수, 8바이트) : ???
boolean(블리언, 1바이트) : true, false
char(문자, 2바이트 유니코드) : 'a'

4. 배열의 요소(element) 개수를 가르키는 것은?

2차원 배열 (배열안에 배열)

```
int[][] intArray = { {1, 2, 3},
                     {4, 5, 6},
                     {7, 8, 9} };
intArray[0][2] => 3
intArray.length => 3
intArray[0].length => 3
intArray[0][0].length => error
```

3차원 배열 (배열안에 배열안에 배열)

```
int[][][] intArray = {
{ {1,2,3},
  {4,5,6},
  {7,8,9} },
{ {1,2,3},
  {4,5,6},
  {7,8,9} } };
intArray.length => 2
intArray[1].length => 3
intArray[1][2].length => 3
intArray[0][0][0].length => error
```

5. JVM이 OS에 할당받은 메모리 중 객체가 저장되는 영역은?

JVM의 메모리

- method area : 클래스의 바이너리코드, 메소드, static 멤버 변수
  생성되면 끝까지 가지고 있으므로 static이 많으면 효율이 떨어질 수 있다.
- heap area : 객체, non-static 멤버 변수, 메모리가 가득차면 (out of memory) 에러 발생_무한루프 생성시
- stack area : 메소드 호출 스택 / stack overflow : 중복된 순환참조로 인해 stack이 무한히 쌓일때

6. 동일 클래스내에 같은 메소드명을 가지고 파라미터를 달리하는 다형성의 예는?

오버로딩(oveloading) : 생성자나 메소드의 이름을 동일하게 하고,
파라미터의 개수, 순서, 타입을 달리 해서 하나의 이름으로 기능 호출하는 방법

ex) print란 메소드로 다음을 출력, 생성자명을 단일화시킴

```
class Overloading {
void print(int a) { System.out.println(a); }
void print(String s) { System.out.println(s); }
void print(int a, int b) { System.out.println(a * b); }

public class Test {
  public static void main(String[] args) {
       Overloading ol = new Overloading();
        ol.print(1);
        ol.print("a");
        ol.print(2, 3);
```

7. 다음 접근지정자(access modifier) 중 동일 패키지와 상속관계일때 접근을 허용하는 것은?

1. public
 1) 모든 패키지에서 접근 가능

2. protected
 1) 동일 패키지 or 다른 패키지에 있더라도 상속 관계일때 접근 가능

3. default
 1) 동일 패키지에서 접근 가능

4. private
 1) 클래스 내에서만(캡슐화_Encapsulation : 정보은닉(information hiding)) 접근 가능

8. 다음 중 오버라이딩(overriding)의 성립 조건이 아닌 것은?

가). 두 클래스가 상속 관계에 있어야 한다.
나). 상속받는 측에서 접근제한자가 덜 접근 제한적이어야 한다.
(예) 상위 default > 하위 defalut, protected, public
다). 반환타입이 같아야 한다. (생성자는 오버라이딩X, 메소드 오버라이딩)
라). 메소드명이 동일해야 한다.
마). 파라미터리스트(개수, 순서, 타입)가 동일해야 한다.
=> 상위메소드와 하위메소드가 같은 메소드라는 얘기
바). 상위 클래스의 메소드와 하위클래스의 메소드가 같은 시그니쳐를 가지고 있다.
- method signautre : 접근제한자 반환타입 메소드명(파라미터리스트)

9. 다음 중 참조타입의 형변환(Type Casting)에 대한 설명으로 옳지 않은 것은?

- 하위타입을 상위타입으로 형변환 할 때는 자동형변환 (상위형변환, 묵시적형변환, implict type casting)
- 상위타입을 하위타입으로 형변환 할 때는 강제형변환 (하위형변환, 명시적형변환, explicit type casting)
- 하위형변환을 하려면 한번은 상위형변환이 된 경험이 있어야 함.

※ 하위로 갈 수록 참조범위가 넓어짐

10. 다음 중 인터페이스에 대한 설명으로 옳지 않은 것은?

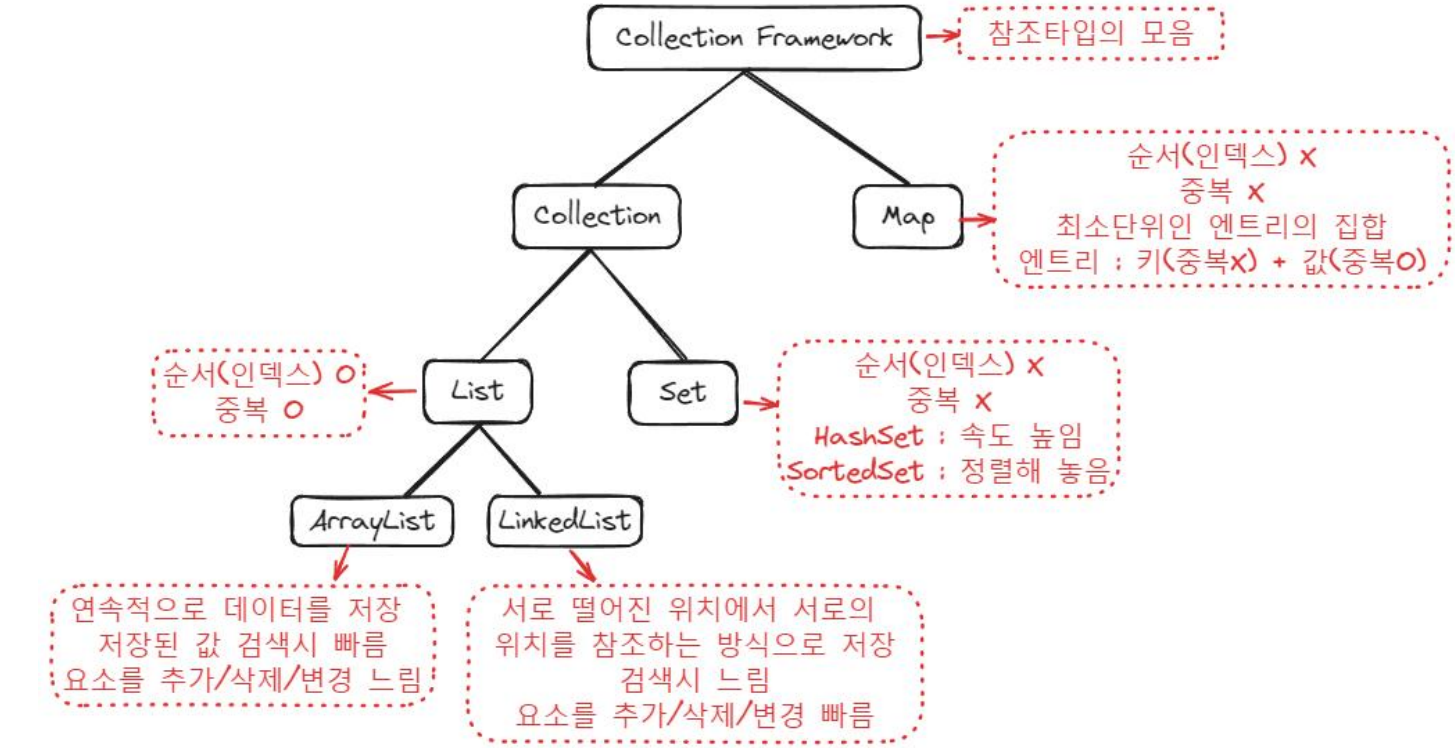
- 설계(Class는 구현, interface는 설계)
- 서로 다른 개체들간의 연결을 위해 존재하는 모든 것들_연결하면서 역할이 있음
- 연결(Connection), 역할(Role) : 서로를 연결해서 어떤 역할을 하게 만든 모든 것들.
서로 다른 2개를 연결하기 위해 있는것들
즉 무엇과 무엇을 연결하고 어떤역할을 하는지를 보는것이 인터페이스의 의의.
- 인터페이스는 하위 클래스들의 설계에 해당하는 것
- 인터페이스에는 생성자가 당연히 없고, 객체 생성이 불가능하다.
- 인터페이스는 인터페이스를 상속받을 수 있다. (상속 : extends / 구현 : implements)
- 클래스는 인터페이스를 구현할 수 있다.(인터페이스가 위에 있음 , 설계이니깐)
- 클래스는 클래스를 상속 받을 수 있다.(단일상속만 가능 - 충돌의 위험성이 있음.)

11. 다음 중 예외처리와 관련하여 옳지 않은 것은?

1) try {} : 예외발생가능한 코드 블록
2) catch (예외클래스) {} : 예외처리 블록, 1개 이상
3) finally {}
: 예외발생여부에 상관없이 무조건 수행되는 코드블록
: 예) 예외발생해도 데이터베이스 커넥션을 닫아야 하는 경우

* 예외처리는 catch 구문에서 일어나는데 하위익셉션>상위익셉션 순으로 catch 구문을 작성해야 한다.
* 예외처리 원칙은 try구문에서 발생할 수 있는 모든 예외에 대해 각각 catch절에서 처리해 주는 것
* throws : 호출한 쪽에 예외처리를 넘김
* throw : 예외를 강제로 발생시킬때 사용하는 키워드

12. 다음 중 컬렉션 프레임워크(Collection Framwork)에 해당하지 않는 것은?



13. 다음 중 쓰레드 클래스를 생성하는 방법이 아닌 것은?

* 쓰레드 생성법
  1) Runnable 인터페이스를 구현
   class MyThread implements Runnable { }
  2) Thread 클래스를 상속
   class MyThread extends Thread { }

14. 다음 중 바이트스트림(Byte Stream)을 읽어 들이기 위한 스트림은?

InputStream : 프로그램으로 들어오는 Byte데이터의 흐름
  * Byte단위 또는 Byte[]단위로 Byte들을 읽는 스트림
OutputStream : 프로그램에서 나가는 Byte데이터의 흐름
  * Byte단위 또는 Byte[]단위로 Byte들을 쓰는 스트림

15. 다음 중 객체직렬화(Object Serialization)에 대한 설명으로 틀린 것은?

  * 직렬화 (Serialization) : 메모리(객체) > 스트림 - 객체를 전송하기 위해 스트림으로 변환하는 작업
  * 역직렬화 (Deserialization) : 스트림 > 메모리(객체) - 전송받은 스트림을 객체로 변환하는 작업

16. (주관식) 기본타입의 형변환에서 가장 중요한 체크포인트는 ( )의 손실여부이다.

* 기본타입의 형변환
 - 형변환이란 데이터가 다른 타입으로 변환되는 것 (byte b = 100 -> int로)
 - 형변환에서 가장 핵심은 데이터의 손실(loss) 여부
 - 표현범위가 작은 타입에서 표현범위가 큰 타입으로의 형변환은 데이터의 손실 가능성이 없다.
   그러므로 묵시적(자동) 형변환
 - 표현범위가 큰 타입에서 표현범위가 작은 타입으로의 형변환은 데이터의 손실 가능성이 있다.
   그러므로 명시적(강제) 형변환

17. (주관식) 객체지향의 3대 개념 중 동일한 형태로 다양한 기능을 구현하기 위한 것은?

객채지향 3대 개념 : 상속, 추상화, 다형성
 1) 상속 : 상위 클래스(Super, Parent)의 변수와 메소드를 하위 클래스(sub, child)에서 상속받아 사용하는 것
    (A extends B : A가 B의 클래스를 상속받는다)
 2) 추상화 : 본연의 성질을 잃어버리지 않는 선에서 최대한 단순화 즉, 대부분의 사람들이 같은 생각이 들게끔 단순화시킬수록 추상화 단계가 높다.
3) 다형성 : 하나의 형태로 다양한 성질을 갖도록 하는 개념.(ex. 오버로딩, 오버라이딩_상속관계에 있는 경우)

18. (주관식) 클래스 앞에 사용하면 상속 불가, 메소드 앞에 사용하면 오버라이딩 불가, 변수 앞에 사용하면 변경 불기

* final 키워드
 - 클래스 앞에 final : 상속 불가
 - 메소드 앞에 fianl : 오버라이딩 불가
 - 변수 앞에 final : 상수(constant, 한번 값이 정해지면 변경할 수 없는 변수)
   상수명은 각 단어를 대문자로 _로 붙인다.
   상수는 반드시 초기화하고 사용한다.

19. (주관식) 객체지향프로그래밍에서 반 설계, 반 구현의 역할을 담당하며 보통 인터페이스와 클래스 중간에 위치하

추상클래스 (Abstract Class)
 - 반설계, 반구현
 - 아직까지 설계대로 전체 메소드(기능)를 다 구현할 수 없는 경우
 - 일반클래스(구체클래스 = concreate class)에서 추상클래스를 상속받아
   추상클래스에서 미구현한 메소들들을 모두 구현하여 객체를 생성
 - 만약에 클래스에 하나라도 추상메소드가 있으면 추상클래스여야 한다. 즉, 객체 생성 불가

20. (주관식) 객체을 바이트 단위로 읽어들이기 위한 Java I/O의 스트림클래스 명은?

DataInputStream / DataOutputStream
  * 기본타입 값들을 읽고 쓰는데 사용됨
  * 기본타입의 바이트수를 지켜서 쓰고 읽어야 한다.

ObjectInputStream / ObjectOutputStream
  * Object 또는 Data를 Byte형태로 변환하는 기술 - ObjectOutputStream(객체 직렬화)
  * Byte를 Object 또는 Data로 변환하는 기술 - ObjectInputStream(객체 역직렬화)

1. 참조타입(reference type)의 형변환에 대해 서술하시오.

참조타입이 형변화를 하려면 상속관계에 있어야 가능하다.
부모클래스와 이를 상속받는 자식클래스가 있다고 가정할 때 하위타입으로 갈수록 참조범위는 넓어진다.
즉 자식클래스에서 부모클래스로 형변환(상위형변환)는 부모클래스보다 자식클래스가 참조범위가 넓음으로 묵시적형변환(자동)
반대로 부모클래스에서 자식클래스로 형변환(하위형변환)은 부모클래스가 자식클래스보다 참조범위가 좁음으로 명시적형변환(강제)
다만 하위형변환을 하려면 한번은 상위형변환이 된 경험이 있어야 한다.

2. 객체지향의 3대개념에 대해 서술하고 예를 보이시오

객채지향 3대 개념 : 상속, 추상화, 다형성

1) 상속 : 상위 클래스(Super, Parent)의 변수와 메소드를 하위 클래스(sub, child)에서 상속받아 사용하는 것
(예제) A extends B : A가 B의 클래스를 상속받는다)

2) 추상화 : 본연의 성질을 잃어버리지 않는 선에서 최대한 단순화 즉
대부분의 사람들이 같은 생각이 들게끔 하는것 / 단순화시킬수록 추상화 단계가 높다.

3) 다형성 : 하나의 형태로 다양한 성질을 갖도록 하는 개념.(ex. 오버로딩, 오버라이딩_상속관계에 있는 경우)

오버로딩(oveloading) : 생성자나 메소드의 이름을 동일하게 하고
파라미터의 개수, 순서, 타입을 달리 해서 하나의 이름으로 기능 호출하는 방법

ex) print란 메소드로 다음을 출력, 생성자명을 단일화시킴

```
class Overloading {
void print(int a) { System.out.println(a); }
void print(String s) { System.out.println(s); }
void print(int a, int b) { System.out.println(a * b); }

public class Test {
  public static void main(String[] args) {
      Overloading ol = new Overloading();
       ol.print(1);
       ol.print("a");
       ol.print(2, 3);
```

추상화 겸 다형성의 오버라이딩 메소드

오버라이딩 : 메소드 재정의_상위클래스에서 하위클래스를 호출 - 호출의 단일화(부모에게 상속받은 메소드를 자식이 재정의 하여 사용)

rrayLotto.java | Animal.java | Cat.java | Dog.java | AnimalMain.java

```
package javabasic;

public abstract class Animal {

    String name;
    int age;

    public Animal(String name, int age) {
        this.name = name;
        this.age = age;
    }

    public abstract void bark(); // 짖는 소리는 동물마다 다름으로 추상메소드 생성
```

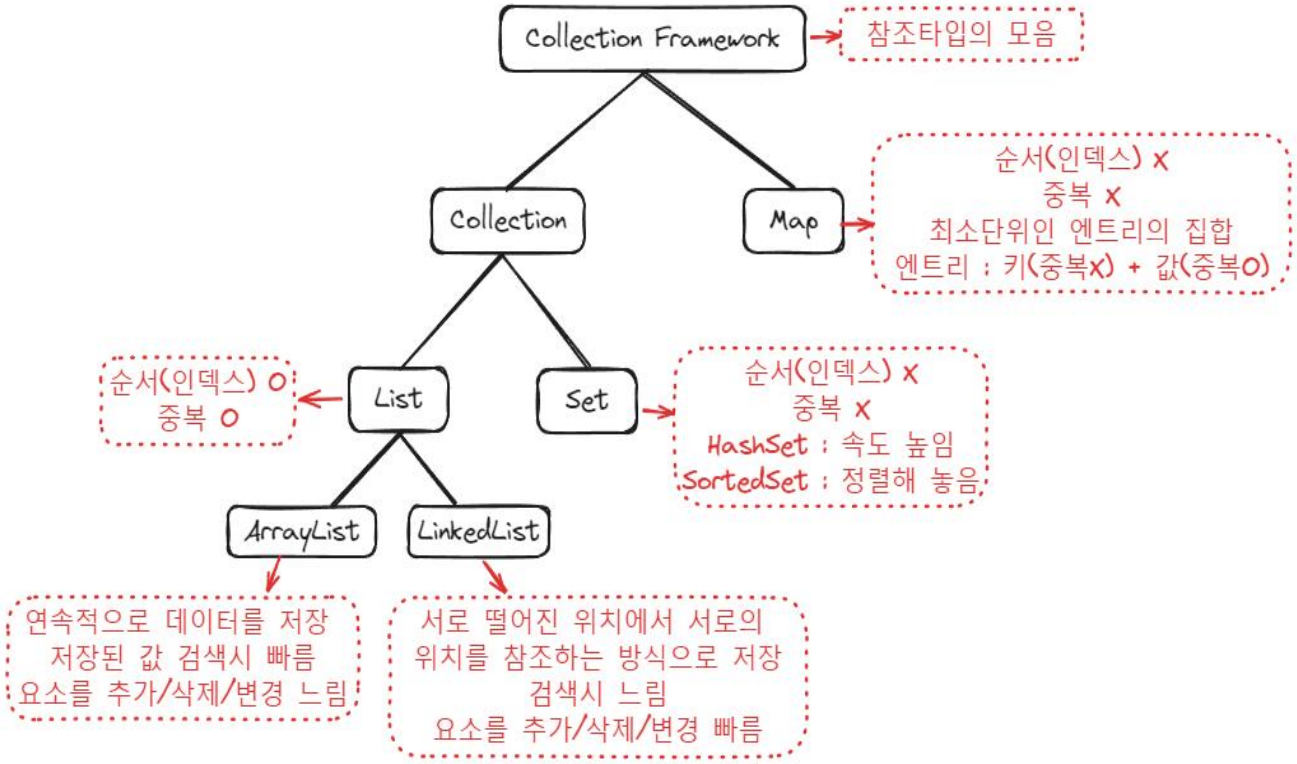
rayLotto.java | Animal.java | Cat.java

```
package javabasic;

public class Cat extends Animal {

    public Cat(String name, int age) {
        super(name, age);
    }

    @Override
    public void bark() {
        System.out.println("야옹!");

    }

}
```

rayLotto.java | Animal.java | AnimalMain.java

```
package javabasic;

public class AnimalMain {

    public static void main(String[] args) {

        Dog dog = new Dog("진돗개", 1);
        Cat cat = new Cat("스핑크스", 3);

        dog.bark();

        cat.bark();

    }

}
```

3. 컬렉션프레임워크(Collection Framework)에 대해 서술하시오.



4. 객체 직렬화(Serialization)과 역직렬화(Deserialization)에 대해 서술하시오.

ObjectInputStream / ObjectOutputStream

* Object 또는 Data를 Byte형태로 변환하는 기술 - ObjectOutputStream(객체 직렬화)
  메모리(객체) -> 스트림 : 메모리내에 있는 객체를 전송하기 위해 스트림으로 변환하는 작업
* Byte를 Object 또는 Data로 변환하는 기술 - ObjectInputStream(객체 역직렬화)
  스트림 -> 메모리(객체) : 전송받은 스트림을 객체로 변환하는 작업

- 직렬화 하려면 Serializable 인터페이스를 구현하도록 강제되어 있음
  - Serializable 인터페이스는 내용이 없다.
    단지 Serializable인터페이스를 구현한 클래스의 객체가 직렬화 가능함을 표시하는 용도 (표시인터페이스)
  - 직렬화 대상
    1) 기본타입
    2) String
    3) private이 아닌 것들
    4) Serializable 인터페이스를 구현한 클래스의 객체