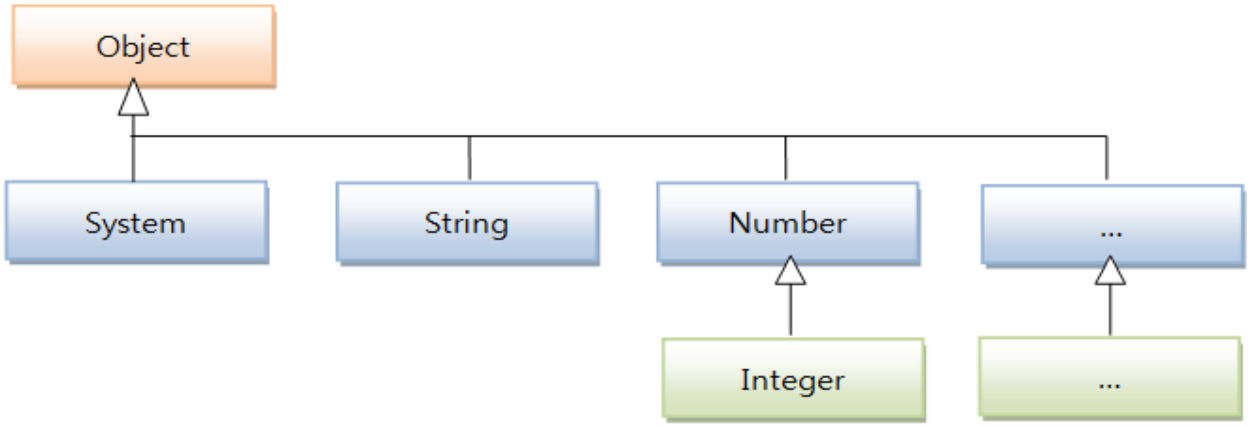
Object 클래스

Object 클래스

❖ Object 클래스

- 자바의 최상위 부모 클래스
- 다른 클래스 상속하지 않으면 java.lang.Object 클래스 상속 암시
- Object의 메소드는 모든 클래스에서 사용 가능



Object 클래스

❖ 객체 비교(equals() 메소드)

```
public boolean equals(Object obj)
```

- Object 클래스의 equlas 메서드는 == 연산자와 동일
  - 물리적 동등성 비교(번지 비교)

```
Object obj1 = new Object();
Object obj2 = new Object();

boolean result = obj1.equals(obj2);
                 기준 객체    비교 객체

boolean result = (obj1 == obj2)
```

결과가 동일

- 논리적 동등성 비교 시 재정의 필요
  - 물리적으로는 다른 인스턴스이지만 가지는 값이 동일한지 여부 판단

Object 클래스

❖ 논리적 동등성 비교 : Member.java

```
public class Member {
    public String id;

    public Member(String id) {
        this.id = id;
    }

    @Override
    public boolean equals(Object obj) {
        if (obj instanceof Member) {          // 매개값이 Member 타입인지 확인
            Member member = (Member) obj;     // 타입변환후
            if (id.equals(member.id)) {       // id 필드 동일 여부 확인
                return true;
            }
        }
        return false;
    }
}
```

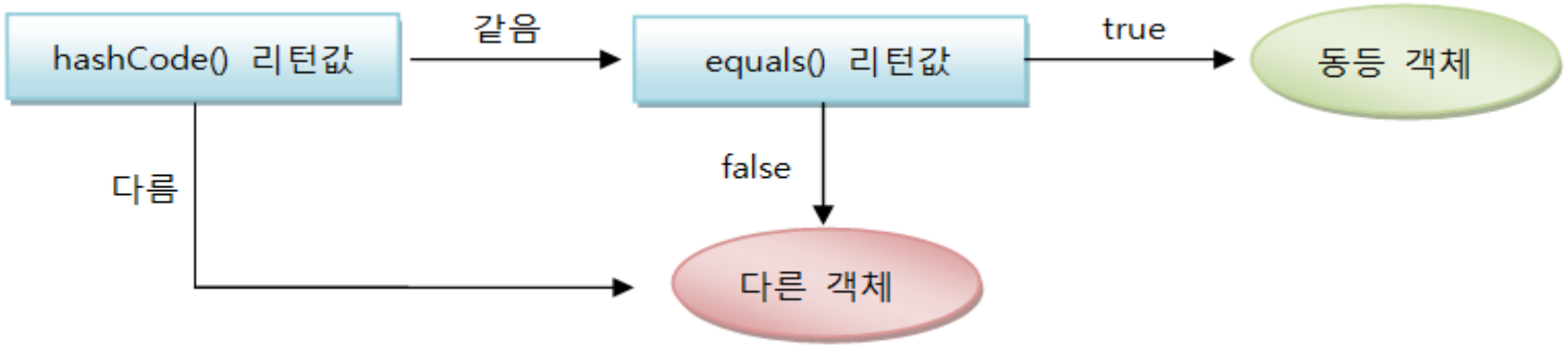
❖ 논리적 동등성 비교 : MemberExample.java

```
public class MemberExample {
    public static void main(String[] args) {
        Member obj1 = new Member("blue");
        Member obj2 = new Member("blue");
        Member obj3 = new Member("red");

        if (obj1.equals(obj2)) {
            System.out.println("obj1과 obj2는 동등합니다.");
        } else {
            System.out.println("obj1과 obj2는 동등하지 않습니다.");
        }

        if (obj1.equals(obj3)) {
            System.out.println("obj1과 obj3은 동등합니다.");
        } else {
            System.out.println("obj1과 obj3은 동등하지 않습니다.");
        }
    }
}
```

Object 클래스

❖ 객체 해시코드(hasCode())

- 객체를 식별할 하나의 정수값을 리턴
  - 디폴트는 객체의 메모리 번지 이용해 해시코드 리턴
    - 개별 객체는 해시코드가 모두 다름
- 논리적 동등 비교시 hashCode() 재정의 필요
  - 컬렉션의 Set, Map
    - 객체(또는 Key)의 중복 저장 불허
    - 논리적 동등성으로 중복 여부 판단



Object 클래스

❖ hashCode() 재정의 : Key.java

```
public class Key {
    public int number;

    public Key(int number) {
        this.number = number;
    }

    @Override
    public boolean equals(Object obj) {
        if(obj instanceof Key) {
            Key compareKey = (Key) obj;
            if(this.number == compareKey.number) {
                return true;
            }
        }
        return false;
    }

    // @Override
    // public int hashCode() {
    //     return number;
    //}
}
```

❖ hashCode() 재정의 객체 비교: KeyExample.java

```
import java.util.HashMap;

public class KeyExample {
    public static void main(String[] args) {
        //Key 객체를 식별키로 사용해서 String 값을 저장하는 HashMap 객체 생성
        HashMap<Key, String> hashMap = new HashMap<Key, String>();

        //식별키 "new Key(1)" 로 "홍길동"을 저장함
        hashMap.put(new Key(1), "홍길동");

        //식별키 "new Key(1)" 로 "홍길동"을 읽어옴
        String value  = hashMap.get(new Key(1));
        System.out.println(value);

        Object obj = new Object();
        System.out.println(obj);
        System.out.println(obj.hashCode());
    }
}
```

Object 클래스

❖ 객체 문자 정보(toString())

- 객체를 문자열로 표현한 값
- String 타입이 지정한 곳에 객체 인스턴스를 배치시 자동 호출
- Object의 디폴트 toString()
  - 클래스명@해시코드(인스턴스주소)

```
Object obj = new Object();
System.out.println( obj.toString() );

[실행 결과]
java.lang.Object@de6ced
```

- 일반적으로 클래스의 필드의 값을 출력
  - 해당 인스턴스의 내부 정보 확인용

Object 클래스

❖ 객체의 내부 정보 출력 : ToStringExample.java

```
import java.util.Date;

public class ToStringExample {
    public static void main(String[] args) {
        Object obj1 = new Object();
        Date obj2 = new Date();

        System.out.println(obj1.toString());
        System.out.println(obj2.toString());
    }
}
```

❖ toString() 재정의 : SmartPhone.java

```
public class SmartPhone {
    private String company;
    private String os;

    public SmartPhone(String company, String os) {
        this.company = company;
        this.os = os;
    }

    @Override
    public String toString() {   // toString() 메서드 재정의
        return company + ", " + os;
    }
}
```

Object 클래스

❖ toString() 재정의 : SmartPhoneExample.java

```
public class SmartPhoneExample {

    public static void main(String[] args) {
        SmartPhone myPhone = new SmartPhone("구글", "안드로이드");

        String strObj = myPhone.toString();
        System.out.println(strObj);

        System.out.println(myPhone);     // myPhone.toString() 자동 호출
    }
}
```

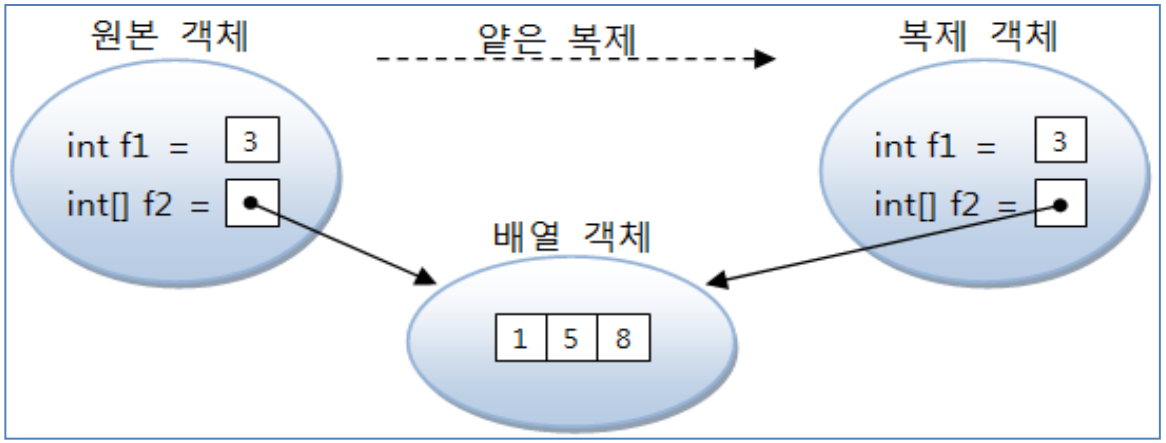
Object 클래스

❖ 객체 복제(clone())

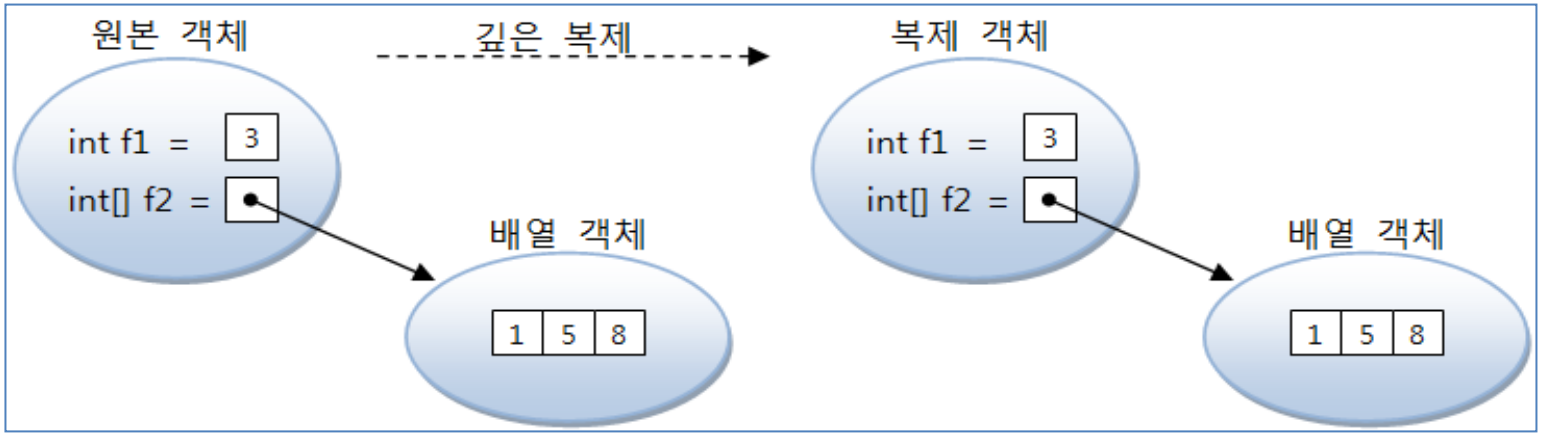
- 원본 객체의 필드 값과 동일한 값을 가지는 새로운 객체를 생성
- 복제의 종류
  - 얕은 복제(thin clone)
    - 필드 값만 복제(기본 타입은 값이 복사, 참조 타입은 참조 값이 복사-공유)
  - 깊은 복제(deep clone)
    - 참조의 대상까지 복제

❖ 객체 복제(clone())

- ㅇ 얕은 복제



- ㅇ 깊은 복제



Object 클래스

❖ 객체 복제(clone())

- Object의 clone() 메서드
  - 동일한 필드 갑을 가지는 얕은 복제된 객체 리턴
    - 따로 clone() 메서드를 재정의할 필요는 없음
  - java.lang.Cloneable 인터페이스 구현 객체만 복제 가능
    - 인텊이스 구현없이 clone() 호출시 CloneNotSupportedException 발생
    - clone() 메서드 호출시 예외처리 필수

```
try {
   Object obj = clone();
} catch(CloneNotSupportedException e) {
   // 예외 처리
}
```

  - 깊은 복제가 필요한 경우 clone() 메서드 재정의
    - 참조 객체도 복제

❖ 복제할 수 있는 클래스 정의 : Member.java

```
public class Member implements Cloneable {
    public String id;
    public String name;
    public String password;
    public int age;
    public boolean adult;

    public Member(String id, String name, String password,
                  int age, boolean adult) {
        this.id = id;
        this.name = name;
        this.password = password;
        this.age = age;
        this.adult = adult;
    }

    public Member getMember() {
        Member cloned = null;
        try {
            cloned = (Member) clone();    // 형변환 필요
        } catch (CloneNotSupportedException e) {
            e.printStackTrace();
        }
        return cloned;
    }
}
```

❖ 복제 객체를 변경하더라도 원본 객체는 변화없음: MemberExample.java

```
public class MemberExample {
    public static void main(String[] args) {
        // 원본 객체 생성
        Member original = new Member("blue", "홍길동", "12345", 25, true);

        // 복제 객체를 얻은 후에 패스워드 변경
        Member cloned = original.getMember();
        cloned.password = "67890";

        System.out.println("[복제 객체의 필드값]");
        System.out.println("id: " + cloned.id);
        System.out.println("name: " + cloned.name);
        System.out.println("password: " + cloned.password);
        System.out.println("age: " + cloned.age);
        System.out.println("adult: " + cloned.adult);
        System.out.println();

        System.out.println("[원본 객체의 필드값]");
        System.out.println("id: " + original.id);
        System.out.println("name: " + original.name);
        System.out.println("password: " + original.password);
        System.out.println("age: " + original.age);
        System.out.println("adult: " + original.adult);
    }
}
```

Object 클래스

❖ 객체 소멸자(finalize())

- GC는 객체를 소멸하기 직전 객체 소멸자(finalize()) 실행
- Object의 finalize() 는 기본적으로 실행 내용이 없음
- 객체가 소멸되기 전에 실행할 코드가 있다면?
  - Object의 finalize() 재정의
- 될 수 있으면 소멸자는 사용하지 말 것
  - GC는 메모리의 모든 쓰레기 객체를 소멸하지 않음
  - GC의 구동 시점이 일정하지 않음