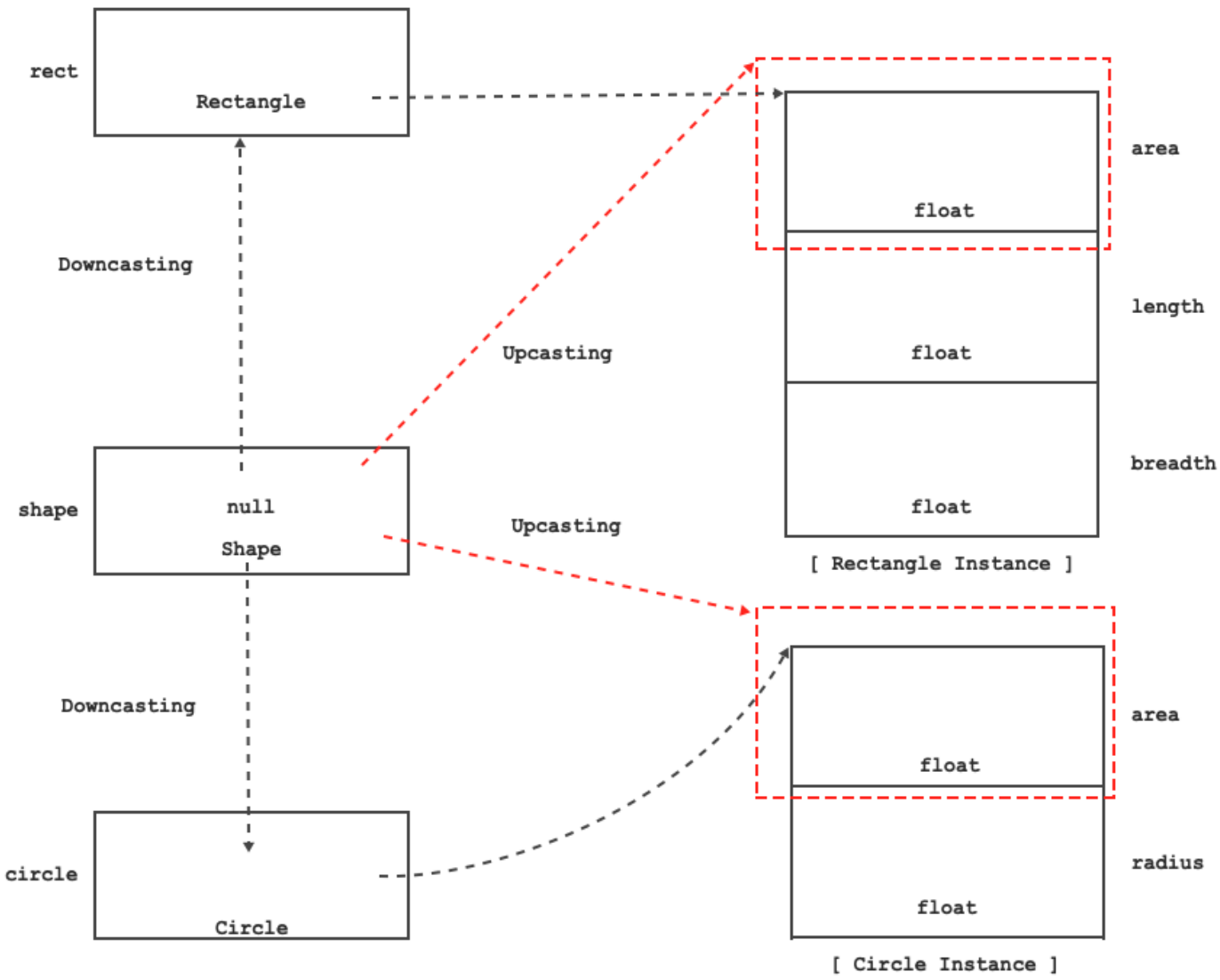
Hierarchy

- If we want to reduce object/instance dependency in the code then we should use upcasting.
- In case of upcasting, using super class reference variable, we can access :
    1. Fields of super class(if it is non private)
    2. Methods of super class
    3. Overriden method of sub class
- In case of upcasting, if we want to access fields of sub class and non overriden method of sub class then we should do downcasting.

instanceof



- instanceof is operator in Java which returns boolean value(true / false)
- If we want to check type of instance at runtime then we should use instance of operator. In other words, we want to check whether super class reference is containing reference of Rectangle instance of Circle instance then we should use instanceof operator.
- Consider example:

```
public void acceptRecord( ) {
    if( this.shape != null ) {
        if( this.shape instanceof  Rectangle ) {
            Rectangle rect =  (Rectangle) this.shape;   //Downcasting
            System.out.print("Length    :   ");
            rect.setLength( sc.nextFloat());
            System.out.print("Breadth   :   ");
            rect.setBreadth(sc.nextFloat());
            //this.shape.calculateArea();
        }else {
            Circle circle = (Circle) this.shape;    //Downcasting
            System.out.print("Radius    :   ");
            circle.setRadius( sc.nextFloat());
            //this.shape.calculateArea();
        }
        this.shape.calculateArea(); //Dynamic method dispatch
    }
}
```

- process of calling method of sub class on reference of super class is called as dynamic method dispatch.

Boxing:

- Process of converting value of primitive type into non primitive type is called as boxing.
- For example:

```
int number = 10;
String str =  String.valueOf( number ); //Boxing
```

```
int number = 10;
Integer i = Integer.valueOf( number );  //Boxing
```

- java.lang.Object is super class of all the classes in Java programming language.
- Consider following code:

```
Integer i1 = new Integer( 123 );
Object o1 = new Integer( 123 ); //Upcasting
Object o2 = Integer.valueOf( 123 ); //Upcasting
```

```
public static void main(String[] args) {
  int number = 123;
  Object o = number;
  // Integer.valueOf(number); //Auto-Boxing
  //Object o = Integer.valueOf(number);   //Upcasting
  System.out.println( o );    //123
}
```

- If boxing is done implicitly then it is called as auto-boxing.

ArrayStoreException:

- If we try to store incompatible value inside array then JVM throws ArrayStoreException
- Consider following code:

```
public static void main(String[] args) {
/* Object o1 = new String("ABC");
Object o2 = new String("PQR");
Object o3 = new String("XYZ"); */

Object[] arr = new String[ 3 ];
arr[ 0 ] =  new String("ABC");
arr[ 1 ] =  "PQR";
arr[ 2 ] =  new StringBuffer("XYZ"); //ArrayStoreException
System.out.println(Arrays.toString(arr));
}
```

What is the difference between operator == and equals.

- We can not compare state of variable of primitive type using equals method.

```
public static void main(String[] args) {
  int num1 = 10;
  int num2 = 10;
  if( num1.equals( num2 ) )   //NOT OK: Cannot invoke equals(int) on the primitive type int
    System.out.println("Equal");
  else
    System.out.println("Not Equal");
}
```

- If we want to compare state/value of variable of primitive type then we should use operator ==.

```
  public static void main(String[] args) {
  int num1 = 10;
  int num2 = 10;
  if( num1 == num2 )  //OK
```

```
    System.out.println("Equal");
  else
    System.out.println("Not Equal");
  //Output: Equal
}
```

- We can use operator == with the variable of non primitive type.

```
  public static void main(String[] args) {
  Employee emp1 = new Employee("Sandeep", 3778, 45000.50f );
  Employee emp2 = new Employee("Sandeep", 3778, 45000.50f );
  if( emp1 == emp2 )  //OK: Comparing state of references
    System.out.println("Equal");
  else
    System.out.println("Not Equal");
  //Not Equal
}
```

- If we want to compare state of references then we should use operator ==.
- If we want to compare state of instances then we should use equals method.
- equals is non final method of java.lang.Object class.
- Syntax:

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

- If we do not define equals() method inside class then super class's equals method will call.
- Consider implementation of equals method from java.lang.Object class:

```
  public boolean equals(Object obj) {
      return (this == obj);
  }
```

```
class Employee{
  private String name;
  private int empid;
  private float salary;
  public Employee(String name, int empid, float salary) {
    this.name = name;
    this.empid = empid;
    this.salary = salary;
  }
}
public class Program {
  public static void main(String[] args) {
    Employee emp1 = new Employee("Sandeep", 3778, 45000.50f );
    Employee emp2 = new Employee("Sandeep", 3778, 45000.50f );
    if( emp1.equals(emp2) )
      System.out.println("Equal");
    else
      System.out.println("Not Equal");
    //Not Equal
  }
  public static void main1(String[] args) {
    Employee emp1 = new Employee("Sandeep", 3778, 45000.50f );
    Employee emp2 = new Employee("Sandeep", 3778, 45000.50f );
    if( emp1 == emp2 )    //OK: Comparing state of references
      System.out.println("Equal");
    else
      System.out.println("Not Equal");
    //Not Equal
  }
}
```

- According to clients requirement, If implementation of super class method is logically incomplete then we should redefine / override method in sub class.

```java
class Employee{
  private String name;
  private int empid;
  private float salary;
  public Employee(String name, int empid, float salary) {
    this.name = name;
    this.empid = empid;
    this.salary = salary;
  }
  //Employee this = emp1;
  //Object obj = emp;   //Upcasting
  @Override
  public boolean equals( Object obj ) {
    if( obj != null ) {
      Employee other = (Employee) obj;  //Downcasting
      if( this.empid == other.empid  )
        return true;
    }
    return false;
  }
}
public class Program {
  public static void main(String[] args) {
    Employee emp1 = new Employee("Sandeep", 3778, 45000.50f );
    Employee emp2 = new Employee("Sandeep", 3778, 45000.50f );
    //Employee emp2 = null;
    if( emp1.equals(emp2) )
      System.out.println("Equal");
    else
      System.out.println("Not Equal");
    //Equal
  }
}
```