# CS 61B Spring 2021

# Inheritance

Exam Prep Discussion 4: February 8, 2021

## 1 Athletes

Suppose we have the Person, Athlete, and SoccerPlayer classes defined below.

```
class Person {
    void speakTo(Person other) { System.out.println("kudos"); }
    void watch(SoccerPlayer other) { System.out.println("wow"); }
}

class Athlete extends Person { void speakTo(Person other)   void watch(SoccerPayer other)
    void speakTo(Athlete other) { System.out.println("take notes"); }
    void watch(Athlete other) { System.out.println("game on"); }
}

class SoccerPlayer extends Athlete {
    void speakTo(Athlete other) { System.out.println("respect"); }
    void speakTo(Person other) { System.out.println("hmph"); }
     void watch(SoccerPayer other)   void watch(Athlete other)
}
```

(a) For each line below, write what, if anything, is printed after its execution. Write CE if there is a compiler error and RE if there is a runtime error. If a line errors, continue executing the rest of the lines.

```
Person itai = new Person();

SoccerPlayer shivani = new Person();  CE

Athlete sohum = new SoccerPlayer();

Person jack = new Athlete();

Athlete anjali = new Athlete();

SoccerPlayer chirasree = new SoccerPlayer();

itai.watch(chirasree);  wow

jack.watch(sohum);  CE    jack->Person->watch(SoccerPlayer other),而sohum的static type是Athlete。

itai.speakTo(sohum);  kudos

jack.speakTo(anjali); ~~take notes~~ kudos

```

jack的static type是Person->Person里面有speakTo(Person other)。anjali的static和dynamic type都是Athlete。合适。

jack的dynamic type是Athlete->Athlete自己写了个seapkTo(Athlete other),同时继承了Person的speakTo(Person other)。在compile的时候确定了使用Person的speakTo(Person other),所以这里还是用这个。

21 `anjali.speakTo(chirasree);` take notes

22

23 `sohum.speakTo(itai);` ~~CE~~ hmph

24

25 `chirasree.speakTo((SoccerPlayer) sohum);` respect

26

27 `sohum.watch(itai);` CE

28

29 `sohum.watch((Athlete) itai);` ~~game on ?~~ RE ((Athlete) itai) is not an illegal cast, because after the cast, itai has a static type Athlete which is more specific than its dynamic type, which is not allowed in Java. Therefore, Although it will compile, a runtime error will occur.

30

31 `((Athlete) jack).speakTo(anjali);` take notes

32

33 `((SoccerPlayer) jack).speakTo(chirasree);` ~~take notes ?~~ RE Similar with line 29. It will compile but a runtime error will occur because ((SoccerPlayer) jack) is not an illegal cast.

34

35 `((Person) chirasree).speakTo(itai);` hmph

(b) You may have noticed that `jack.watch(sohum)` produces a compile error. Interestingly, we can resolve this error by **adding casting**! List two fixes that would resolve this error. The first fix should print `wow`. The second fix should print `game on`. Each fix may cast either `jack` or `sohum`.

1. (SoccerPlayer) sohum print: wow
2. (Athlete) jack print: game on

(c) Now let's try resolving as many of the remaining errors from above by **adding or removing casting**! For each error that can be resolved with casting, write the modified function call below. Note that you cannot resolve a compile error by creating a runtime error! Also note that not all, or any, of the errors may be resolved.

line 15 jack.watch(sohum)
改法1：(SoccerPlayer) sohum 打印 wow
改法2：(Athlete) Jack 打印 game on

line 33 ((SoccerPlayer) jack).speakTo(shirasree)
改法1：remove SoccerPlayer cast to Jack 打印 kudos
改法2：(Athlete)Jack 打印speakTo(Athlete other)的take notes（具体见java代码）

## 2 Dynamic Method Selection

Modify the code below so that the max method of DMSList works properly. Assume all numbers inserted into DMSList are positive, and we only insert using `insertFront`. You may not change anything in the given code. You may only fill in blanks. You may not need all blanks. (Spring '16, MT1)

```java
public class DMSList {
    private IntNode sentinel;
    public DMSList() {
        sentinel = new IntNode(-1000, _____________________);
    }
    public class IntNode {
        public int item;
        public IntNode next;
        public IntNode(int i, IntNode h) {
            item = i;
            next = h;
        }
        public int max() {
            return Math.max(item, next.max());
        }
    }
    public _________________________________ {

    ____________________________________________________________

    ____________________________________________________________

    ____________________________________________________________

    ____________________________________________________________

    ____________________________________________________________

    ____________________________________________________________

    ____________________________________________________________

    ____________________________________________________________
    }
    /* Returns 0 if list is empty. Otherwise, returns the max element. */
    public int max() {
        return sentinel.next.max();
    }
    public void insertFront(int x) { sentinel.next = new IntNode(x, sentinel.next); }
}
```

# 3 Challenge: A Puzzle

Consider the **partially** filled classes for A and B as defined below:

```
public class A {
    public static void main(String[] args) {
        _A_ y = new _B_();
        _B_ z = new _B_();
    }

    int fish(A other) {
        return 1;
    }

    int fish(B other) {
        return 2;
    }
}

class B extends A {
    @Override
    int fish(B other) {
        return 3;
    }
}
```

int fish(A other){return 1;}

Note that the only missing pieces of the classes above are static/dynamic types! Fill in the **four** blanks with the appropriate static/dynamic type — A or B — such that the following are true:

1. y.fish(z) equals z.fish(z) 3
2. z.fish(y) equals y.fish(y) 1
3. z.fish(z) does not equal y.fish(y)
   3 1