Lec 19

Inheritance & polymorphism
• Quiz Friday
• Project proposals due on Saturday
public class XXX extends Car
class RX7 extends Car {
    private int NOS;
    public RX7() {
        NOS = 10;
    }

    public void useNOS() {
        System.out.println("hold on");
        NOS--;
        accelerate();
    }
}

class Car {
    private String color;
    public Car() {
        color = "RED";
    }
    public void accelerate() {
        System.out.println("let’s GO");
    }
}

class Start {
    public static void main(...) {
        RX7 dom = new RX7();
        dom.useNOS();
        dom.accelerate();
    }
}
Constructors (reminder)

Purpose: Initialize vars in newly allocated object

They may be overloaded (same name, different signature)

```java
public Foo() { ... }
public Foo(String name) { ... }
```

If a class contains no constructor definitions

Compiler will automatically insert a default (no-arg) constructor
Constructors in inheritance

Constructor body

First line of code may be either

\[
\text{this}(\; \text{args}_{\text{opt}}\; ) \quad - \quad \text{same class ctor call}
\]

or

\[
\text{super}(\; \text{args}_{\text{opt}}\; ) \quad - \quad \text{super class ctor call}
\]
public class Faculty extends Person {
    public Faculty() {
        this("hi");
    }
    public Faculty(String intro) {
        super();
        System.out.println(intro);
    }
}

public class person {
    public Person() {
        System.out.println("person");
    }
}

Faculty adam = new Faculty();

A) hi, person
B) person, hi
C) Compiler error
D) Runtime error
Constructors

public class Point {
    private int x, y;

    public Point() {
    }
}

public class Point extends Object {
    private int x, y;

    public Point() {
        super();
    }
}

compiler automatically inserts super() as first statement in sub’s constructor if the programmer doesn’t call themselves.

Compile error if super class does not have a no-arg ctor defined
Constructors

This is to ensure objects are initialized.

Superclass ctors execute first, then then subclass ctors.
public class Faculty extends Employee {
    public static void main(String[] args) {
        new Faculty();
    }

    public Faculty() {
        super();
        System.out.println("1");
    }
}

class Employee extends Person {
    public Employee() {
        this("2");
        System.out.println("3");
    }

    public Employee(String s) {
        System.out.println(s);
    }
}

class Person {
    public String name;
    public Person() {
        S.O.P(4);
    }
}
public class Faculty extends Employee {
    public static void main(String[] args) {
        new Faculty();
    }

    public Faculty() {
        super("1");
        System.out.println("1");
    }
}

class Employee extends Person {
    public Employee() {
        this("2");
        System.out.println("3");
    }

    public Employee(String s) {
        System.out.println(s);
    }
}

class person {
    public Person() {
        System.out.println("4");
    }
}

A) 1, 2, 1
B) 4, 2, 1
C) 4, 2, 3, 1, 1
D) 4, 1, 1
E) 1, 2, 3, 4, 1
Method Overriding
public class Car {
    public Car() {
        //initialize car
    }

    public void accelerate() {
        //rotate wheels
    }
}

class Tank extends Car {
    public Tank() {
        //initialize tank
    }

    public void accelerate() {
        //rotate track
    }
}
Overridden Methods

To invoke the superclass's version of an overridden method:

```java
public void accelerate() {
    smokeTires();
    super.accelerate();
}
```
public class Car {
    private int damage = 10;
    private String type = "generic";

    public Car (int damage, String type) {
        this.damage = damage;
        this.type = type;
    }

    public void drive() {
        System.out.println(type + " driving");
    }
}

public class Tank extends Car {
    public Tank() {
        super(10, "Tank");
    }

    public void drive() {
        super.drive();
        System.out.println("roll out");
    }
}

Tank t = new Tank();
t.drive();

A) generic + driving, roll out
B) Tank + driving
C) Tank + driving, roll out
D) Run time error
E) Compiler error
public class Car {
    private int damage = 10;
    private String type = “generic”;

    public Car () {} 

    public void accelerate() {
        rotate();
    }
    public void rotate() {
        System.out.println(“tires”);
    }
}

Public class Tank extends Car {
    public Tank() {
    }
    public void rotate() {
        System.out.println(“track”);
    }
}

Tank t = new Tank();
t.accelerate();
public class Animate extends WindowController {

    public void begin() {
        //where you code your animation
    }

    public static void main(String[] args) {
        new Animate().startController(SIZE, SIZE+LINUX_MENU_BAR);
    }
}
Polymorphism

```
Car
  - damage: int
  - color: String
  + Car()
  + Car(damage: int, color: String)
  + accelerate(): void
  + break(): void

FastCar
  + FastCar(damage: int, color: String)

Zamboni
  + Zamboni(damage: int, color: String)

Tank
  - ammo: int
  + Tank(damage: int, color: String, ammo: int)
  + fire(): void
```
public class Character {
    private Car currentCar;

    public void stealCar() {
        currentCar = ???;
    }
}
Declared type vs. actual type

```java
public class Character {
    private Car currentCar;

    public void stealCar() {
        currentCar = new Tank();
    }
}
```
Declared type vs. actual type

Car currentCar = new Car();
Car currentCar = new FastCar();
Car currentCar = new Tank();

Tank currentCar = new Tank();
Tank currentCar = new Car();
Declared type vs. actual type

Car currentCar = new Car(); ★
Car currentCar = new FastCar(); ★
Car currentCar = new Tank(); ★

Tank currentCar = new Tank(); ★
Tank currentCar = new Car(); ☹
Garage

- fastCars: FastCar[]
- zambonis: Zamboni[]
- tanks: Tank[]

+ Garage()
+ addFastCar(c: FastCar): void
+ addZamboni(c: Zamboni): void
+ addTank(c: Tank): void
Garage

- fastCars: FastCar[]
- zambonis: Zamboni[]
- tanks: Tank[]

+ Garage()
+ addFastCar(c: FastCar): void
+ addZamboni(c: Zamboni): void

+ Garage()
+ addCar(newCar: Car): void
Polymorphism – Key Points

4 ways to match super/sub class references with super/sub class objects:

Super class ref -> Super class object
   Super superRef = new Super();  // Same type

Sub class ref -> Sub class object
   Sub subRef = new Sub();        // Same type

Super class ref -> Sub class object
   Super superRef = new Sub();    // Basis for polymorphism
   (Can only access members that are common to both Super and Sub)

Sub class ref -> Super class object
   Sub subRef = new Super();      // Compile Error!
   (Not an is-a relationship. Could try to access Sub class type only members that do not exist in the Super class object!)
public class Car{
    private int damage;
    private String color;
    public Car() {} 
    public Car(int damage, String c) {} 
    public void accelerate() {
        System.out.println("car"); 
    }
}

public class FastCar extends Car {
    public FastCar(int d, String c) {} 
}

public class Zamboni extends Car {
    public Zamboni(int d, String c) {} 
    public void accelerate() {
        System.out.println("ice"); 
    }
}

public class Tank extends Car {
    public Tank(int d, String c, int a) {} 
    public void fire() {
        System.out.println("FIRE"); 
    }
}

Car c = new Tank(0,"Tank", 10); 

What gets printed?
A) Compiler error 
B) Run time error 
C) FIRE
public class Car{
    private int damage;
    private String color;
    public Car() {}
    public Car(int damage, String c) {}
    public void accelerate() {
        System.out.println("car");
    }
}

public class FastCar extends Car {
    public FastCar(int d, String c) {}
}

public class Zamboni extends Car {
    public Zamboni() {}
    public Zamboni(int d, String c) {}
    public void accelerate() {
        System.out.println("ice");
    }
}

public class Tank extends Car {
    public Tank(int d, String c, int a) {}
    public void fire() {
        System.out.println("FIRE");
    }
}

What gets printed?
A) Compiler error
B) Run time error
C) car
D) ice
public class Car{
    private int damage;
    private String color;
    public Car() {}  
    public Car(int damage, String c) {}  
    public void accelerate() {
        System.out.println("car");
    }
}

public class FastCar extends Car {
    public FastCar(int d, String c) {}  
}

public class Zamboni extends Car {
    public Zamboni(int d, String c) {}  
    public void accelerate() {
        System.out.println("ice");
    }
}

public class Tank extends Car {
    public Tank(int d, String c, int a) {}  
    public void fire() {
        System.out.println("FIRE");
    }
}

Car c = new Tank(0,"Tank", 10);
((Tank) c).fire();

What gets printed?
A) Compiler error
B) Run time error
C) FIRE