Lec 11

Inheritance
FastCar

- topSpeed: int
- bodyType: String
- color: String
- damage: int

+ Car()
+ Car(topSpeed: int, bodyType: String, color: String, damage: int)
+ accelerate()
+ reverse()
+ brake()

Zamboni

- topSpeed: int
- bodyType: String
- color: String
- damage: int

+ Car()
+ Car(topSpeed: int, bodyType: String, color: String, damage: int)
+ accelerate()
+ reverse()
+ brake()

Tank

- topSpeed: int
- bodyType: String
- color: String
- damage: int
- ammunition: int

+ Tank()
+ Tank(topSpeed: int, bodyType: String, color: String, damage: int)
+ accelerate()
+ reverse()
+ brake()
+ Fire()
public class XXX extends Car
• Refers to “this instance” of an object
  – Think of as this = object’s memory address
• Used for:
  1. Disambiguates variable references
  2. Call alternate constructors from within a constructor
  3. Pass the current class instance as an argument to a method of another object
public class Student {
    private int grade;
    private char letterGrade;

    public Student() {
        this(100, 'A');
    }

    public Student(int grade, char letter) {
        this.grade = grade;
        letterGrade = letter;
    }

    public void changeGrade(int grade) {
        this.grade = this.grade + grade;
    }

    public void printGrade() {
        System.out.println(grade + " : " + letterGrade);
    }
}

Student s = new Student();
s.changeGrade(-5);
s.printGrade();

A) : 
B) -5 :
C) 100 : ‘A’
D) 95 : ‘A’
E) Compiler error
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

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

or

\[ \text{super}( \text{args}_{opt} ) \] - 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

```java
public class Point {
    private int x, y;

    public Point() {
        // compiler automatically inserts super() as first statement in sub’s constructor
        super();
    }
}
```

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

    public Point() {
        super();
    }
}
```

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() {
        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, 3, 4
B) 4, 3, 2, 1
C) 4, 2, 3, 1
D) 4, 3, 1, 2
E) 1, 3, 2, 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");
    }
}
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();
}
```
What gets printed

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

Public class Tank extends Car {
    public Tank() {
    }
    public void drive() {
        super.drive();
        System.out.println("roll out");
    }
}

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

A) generic + driving, roll out
B) + driving
C) roll out
D) Run time error
E) Compiler error
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
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

```java
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
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");
    }
}

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