CSE 1322
Module 3 – Part 2

Polymorphism

KENNESAW STATE UNIVERSITY

Does this look good?

```
class Mammal{
}

class Dog extends Mammal{
}

class Cat extends Mammal{
}
```

```
Dog d1 = new Dog();
Dog d1 = new Cat();
```

KENNESAW STATE UNIVERSITY

How about this?

```
class Mammal{
}

class Dog extends Mammal{
}

class Cat extends Mammal{
}
```

```
Mammal d1 = new Dog();
Mammal d2 = new Cat();
```

KENNESAW STATE UNIVERSITY

Or this?

```
class Mammal{
}

class Dog extends Mammal{
}

class Cat extends Mammal{
}
```

```
Mammal d1 = new Dog();
d1 = new Cat();
```

KENNESAW STATE UNIVERSITY

Polymorphism

- The first one is clearly an incompatible type error. We cannot create a **Dog** object and initialize it with a **Cat** type object.
- In contrast, the last two are completely valid.
- The last two snippets of code are leveraging **Polymorphism**.

KENNESAW STATE UNIVERSITY

Polymorphism – Etymology

Polymorphism - *Objects that take more than one form*

- Poly – means “many” (πολλές – “polles”)
- Morph – means “forms” (μορφές – “morfes”)

KENNESAW STATE UNIVERSITY

Polymorphism

- Polymorphism allows a superclass reference to refer to objects of different subclasses, enabling dynamic behavior at runtime.

```
Mammal d1 = new Dog();
Mammal d2 = new Cat();
```

Polymorphism

- Since **Mammal** is a superclass for subclasses **Dog** and **Cat**, this code snippet is valid.

```
Mammal d1 = new Dog();
Mammal d2 = new Cat();
```

- Polymorphism further enhances our ability to write flexible and dynamic code.

Polymorphism – Types

- There are two main types of Polymorphism:
  - Compile-Time or Static Binding
  - Run-Time or Late Binding

KENNESAW STATE UNIVERSITY

Compile-Time – Method Overloading

- We have encountered this previously with **Method Overloading**.
- Compile-Time polymorphism occurs when multiple methods had the same **name** a but different **signature**.
- With Static-Binding, the compiler will determine which respective method to call based on the signature defined in the method call.
- Since the compiler determines this, then **everything is resolved at compile time**.

Compile-Time

```
public static float tryMe(int x){
    return x + .375f;
}

public static float tryMe(int x, float y){
    return x * y;
}

public static void main(String[] args) {
    float result = tryMe(25, 4.32f);
}
```

Week-3/Overloaded1.java

KENNESAW STATE UNIVERSITY

Compile-Time – Method Overloading

- Since everything is resolved at compile time, this makes Compile-Time polymorphism not **true** polymorphism.
- **True** polymorphism requires that method resolution be done during runtime.

KENNESAW STATE UNIVERSITY

Run-Time

- As the name implies, Run-Time Polymorphism allows method class to be dynamically resolved at runtime.
- This resolution is based on the **actual object instance** rather than the **reference type**.
- This enables dynamic method dispatch and flexible object behavior.
- Dynamic binding is **true** polymorphism.

KENNESAW STATE UNIVERSITY

Run-Time

- With Run-Time Polymorphism, we can make a reference of a superclass and use this reference to point to any of its subclasses

```
Mammal m1;
m1 = new Dog();
m1 = new Cat();
```

Run-Time – Method Overriding

- Run-Time method overriding happens when a **variable** of the **superclass type** references an **object** of its **subclass**.
- Late Binding polymorphism occurs when a subclass provides a specific implementation of a method that is already defined in its superclass.
- The method call is **determined at runtime based on the actual object type**.

KENNESAW STATE UNIVERSITY

Run-Time – Method Overriding

- At runtime, Java will determine the actual type of the object and invoke the corresponding overridden methods.

```java
Mammal m1;
m1 = new Dog();
m1.Eat();
m1 = new Cat();
m1.Eat();
```

Run-Time – Method Overriding

```java
class Mammal{
    public void Eat(){
        System.out.println("This mammal is eating");
    }
}

class Dog extends Mammal{
    @Override
    public void Eat(){
        System.out.println("This dog is eating");
    }
}

class Cat extends Mammal{
    @Override
    public void Eat(){
        System.out.println("This cat is eating");
    }
}

public class Runtime {
    public static void main(String[] args) {
        Mammal m1;
        m1 = new Dog();
        m1.Eat();
        m1 = new Cat();
        m1.Eat();
    }
}
```

```
This dog is eating
This cat is eating
```

Week-5/Polymorphism/Runtime.java

KENNESAW STATE UNIVERSITY

Inheritance and Polymorphism

- Inheritance allows a subclass to acquire attributes and behaviors from a superclass, therefore **inheritance enables code reuse**.
- Polymorphism allows the same method to behave differently depending on which subclass calls it; therefore, **polymorphism enables flexible and dynamic code**.

KENNESAW STATE UNIVERSITY