DiagClassi: c1—x..yc2 (ogni c1 è legato a x..y c1); {ordered}, {disjoint, complete}, {subset}; freccia sottoclasse verso su;

eventoScatenante(payload) {mitt:..}{dest:..} [condizione]/ eventoConseguente(payload) {mitt:..}{dest:..} (passaggio di stato)

Resp: Molteplicità (lato opposto), Operazioni, Requisiti. Ogni associazione almeno un responsabile

SpecAttCompl: Nome(IN)(OUT); VarAux: tipo; Processo in pseudocodice con sottoattività (nome(in)(out), fork: t1:{att}, t2:{att}, join); FINE SPEC

SpecStatiClasse: STATI:{tutti}; VarAux: tipo; ST.INIZ: statocorrente=s1, var=--; FINE SPEC

SpecTransClasse: Transizione: s1->s2; Evento: ev(payload); Condizione: c; Azione: pre:-- post: nuovoevento=azione, aggiorno variabili;

AssBin Resp sing: 0..1 (classe c=null; get e set); 0..\* (HashSet<classe> nome; Aggiungi, Elimina e Get -add, remove, clone); attributo (creo TipoLinkAssociazione o Set di TLA e metto get, set/add, remove);

AssBin Resp doppia: TLA e MA; TLA link; public void inserisciLA (TLA t){MA.inserisci(t);} / elimina; public void inserisciPerMA (MA a) {link=a.getLink} / elimina con link=null; public TLA getLA(){return link;} SET: nei PerMA ci saranno add e remove ad un set, get restituirà Set;

TipoLinkAssociazione: c1, c2, attributi, costruttore con attributi, get per ogni elemento, equals, hashcode.

ManagerAssociazione: private MA(TLA x){link=x;} private TLA link; public TLA getLink(){return link;} public static void inserisci(TLA y) {MA k=new MA(y); y.getc1().inserisciPerMA(); y.getc2.inserisciPerMA();} / elimina;

boolean equals (Object o) {if (o!=null & getclass().equals(o.getclass())) {C c=(C) o; return x=c.x && y=c.y;} else return false;}

int hashcode() {return super.hashcode()+..}

public class Classe implements Listener: valori da DiagClassi con get e set, costruttore; public statice num Stato {s1, s2}; statocorrente=Stato.s1; public void fired (Evento e) {TaskExecutor.getInstance().perform(new ClasseFired(this, e));} Associazioni;

ClasseFired implements Task (import Classe.Stato): Classe c; Evento e; costruttore con c ed e; public synchronized void esegui(Evento e) {switch(statocorrente){case s1 if(e.getClass()==eventointeressante.class){if(condizione){… statocorrente=Stato.s2; Environment.aggiungiEvento(new Evento(mitt, dest, payload));} break} case s2…}}

Attività principale implements Runnable: in run crea thread e lanciali.

Attività complesse implements Runnable: in run TaskExecutor.getInstance().perform(new AttivitàAtomica()); SegnaliIO.inviaOattendiQualcosa();

Thread.start(new Funtore()); Thread.join();

Set<Integer> s; s = new HashSet<tipo>(); s.add(tipo); s.remove(tipo); return (HashSet<tipo>)s.clone();

SAC: segn, varaux, processo(fork,join); SS: stati, varaux, st.iniz con var; ST: trans, event, cond, az prepost;

Classe impl.List, valorigetset+costr, staticenumstato, stcorr+get, fired (evento) {TaskExecutor.getInstance(). perform (new clfired (this,e))}, 5assoc;

TLA: c1c2+cost+get, eq+hash; MA: tla, privcost, staticinser(TLA) (creaMA, insperma(MA));

ClasseFired impl.Task importStato, cl,ev+cost, syncesegui(Ev e) {if dest=cl switch(getStato) {case S1: if(e.getClass=int.class) cl.stcorr=S2 Environment.addev (new ev (mit,des,payl)) break}}

AttPrin impl.RunSYNC (thread, TaskExec…, SegIO.attendi())